

# Transformer Models

Prof. Giuseppe Serra

University of Udine

Attention Is All You Need (Google)

# Transformer (Google 2017)

- Transformer was developed for solving text translation problem.
- The Transformer architecture consists of an Encoder and a Decoder and the Encoder's output is an input to the Decoder.

## Attention Is All You Need

Ashish Vaswani\*  
Google Brain  
avaswani@google.com

Noam Shazeer\*  
Google Brain  
noam@google.com

Niki Parmar\*  
Google Research  
nikip@google.com

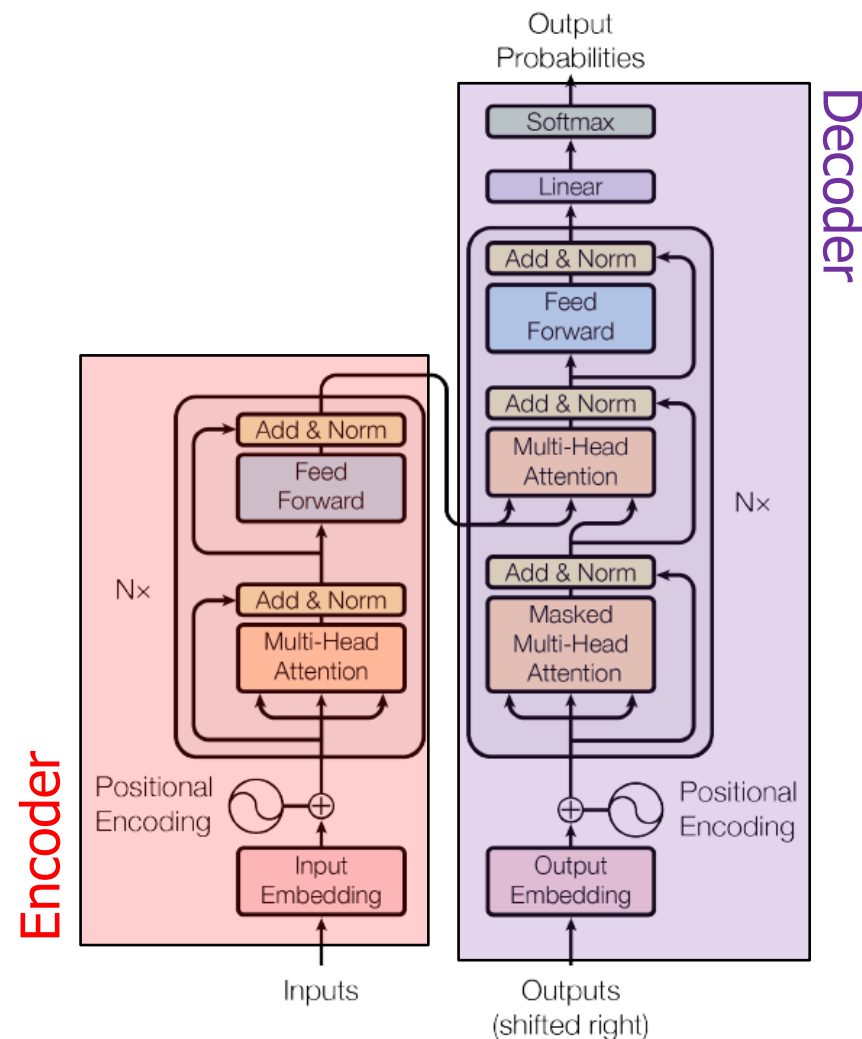
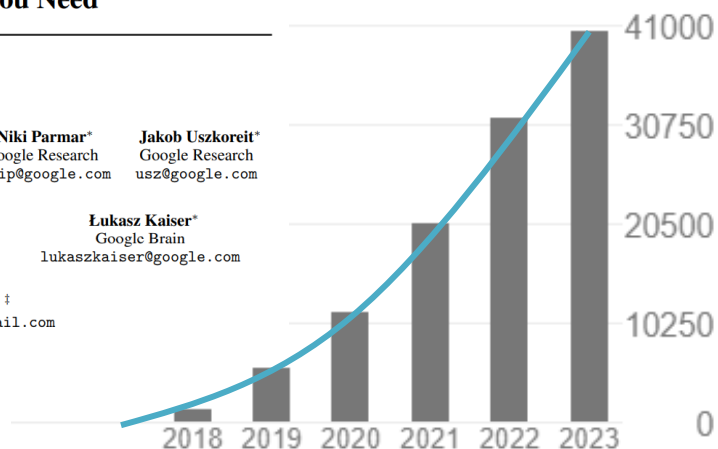
Jakob Uszkoreit\*  
Google Research  
usz@google.com

Llion Jones\*  
Google Research  
llion@google.com

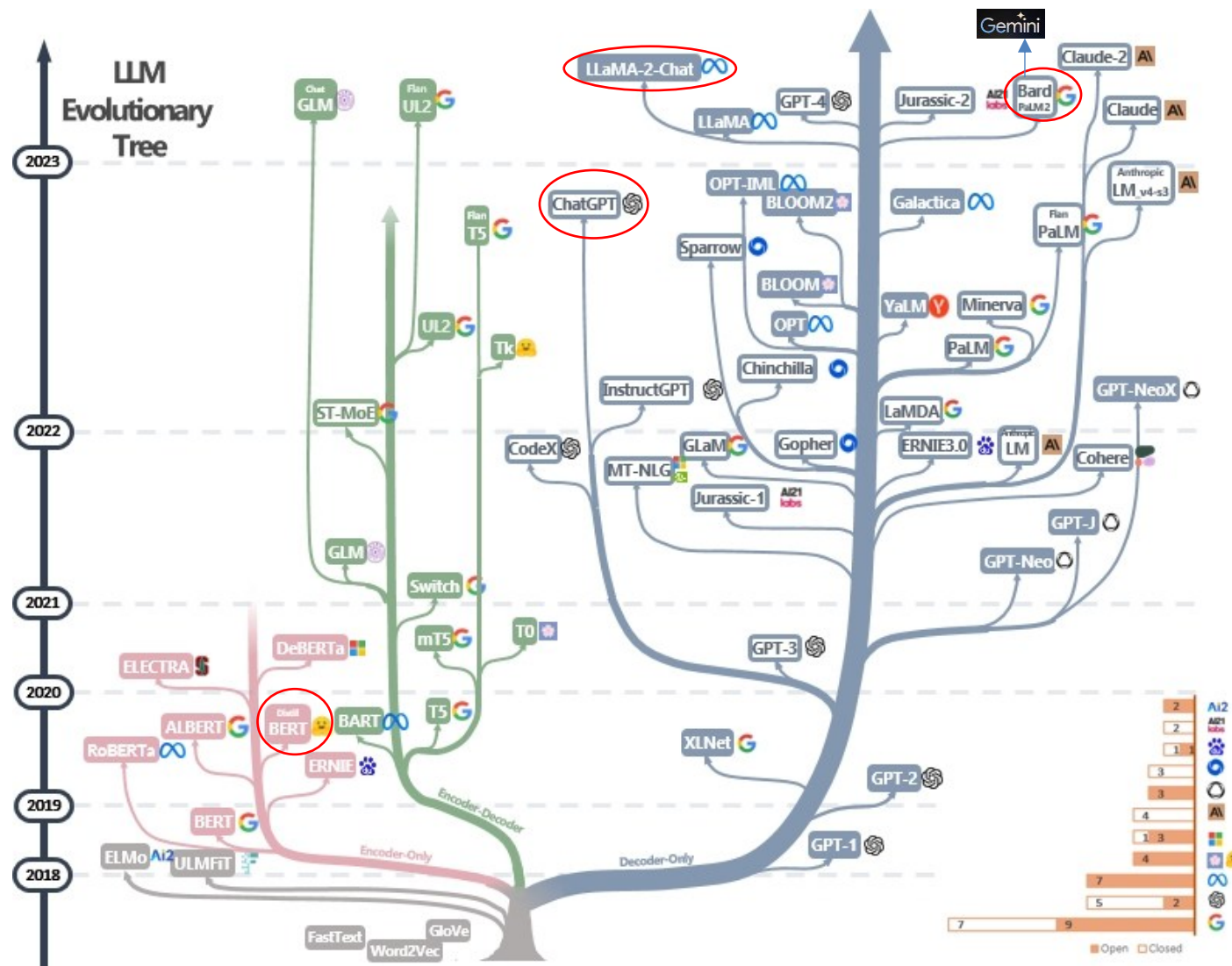
Aidan N. Gomez\*<sup>†</sup>  
University of Toronto  
aidan@cs.toronto.edu

Łukasz Kaiser\*  
Google Brain  
lukaszkaier@google.com

Illia Polosukhin\*<sup>‡</sup>  
illia.polosukhin@gmail.com



# LLM Evolutionary Tree



# BERT: Bidirectional Encoder Representations from Transformers

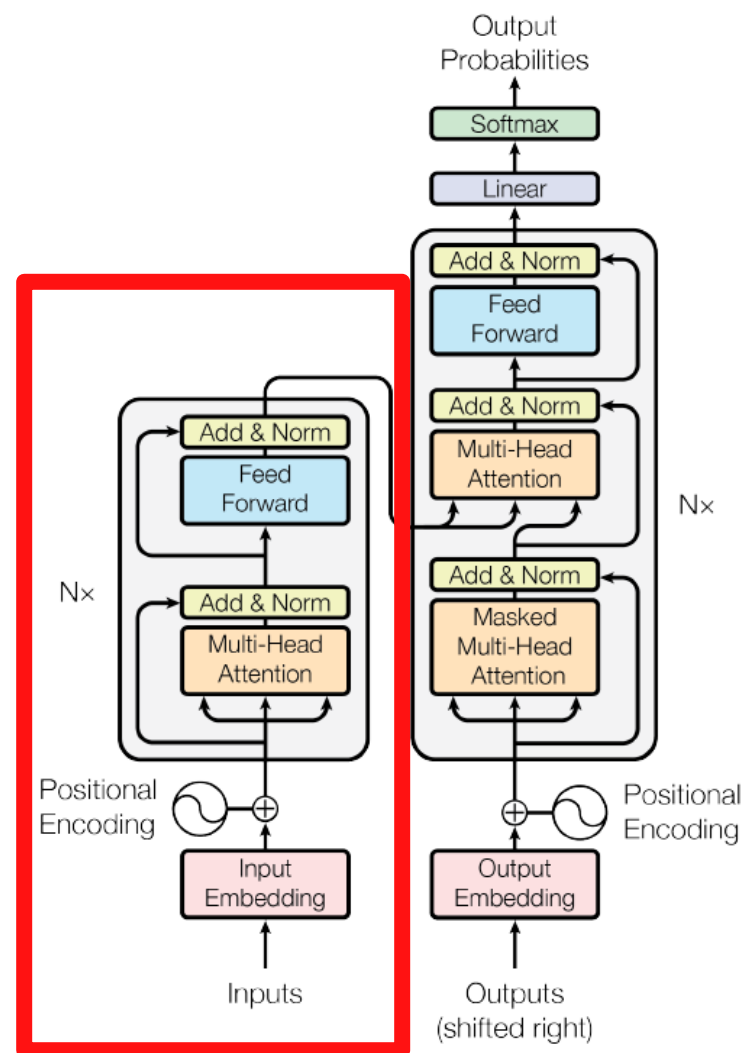
# BERT (Google 2018)

---

- BERT is a paper published by researchers at Google AI Language.
- It has caused a stir in the Machine Learning community by presenting state-of-the-art results in a wide variety of NLP tasks:
  - Question Answering (SQuAD v2.0);
  - Named Entity Recognition
- BERT's key technical innovation is applying the bidirectional training of Transformer, a popular attention model, to language modelling.
- This is in contrast to previous efforts which looked at a text sequence either from left to right or combined left-to-right and right-to-left training.
- The paper's results show that a language model which is bidirectionally trained can have a deeper sense of language context and flow than single-direction language models.

# BERT (Google 2018)

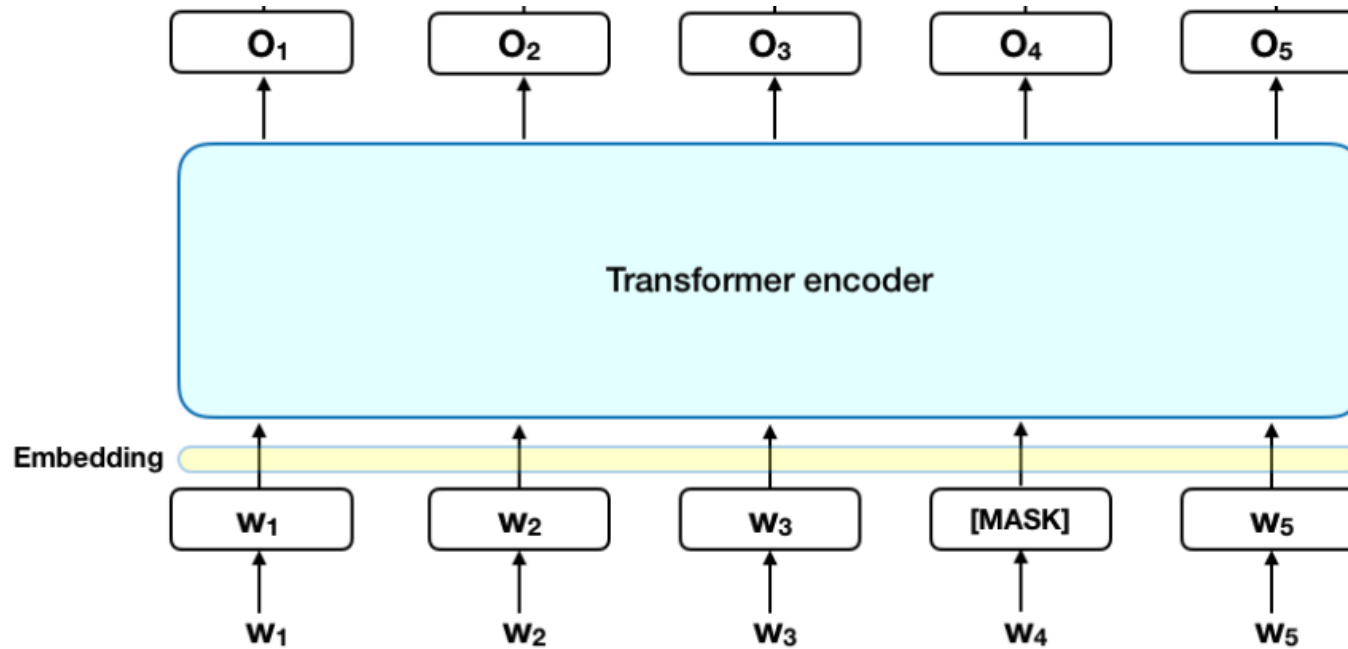
- In short BERT consists of the Encoder, introduced in the paper Attention Is All You Need, with some novel modifications.



# Transformer Encoder

---

- The input is a sequence of tokens, which are first embedded into vectors and then processed in the neural network.
- The output is a sequence of vectors of size  $H$ , in which each vector corresponds to an input token with the same index.





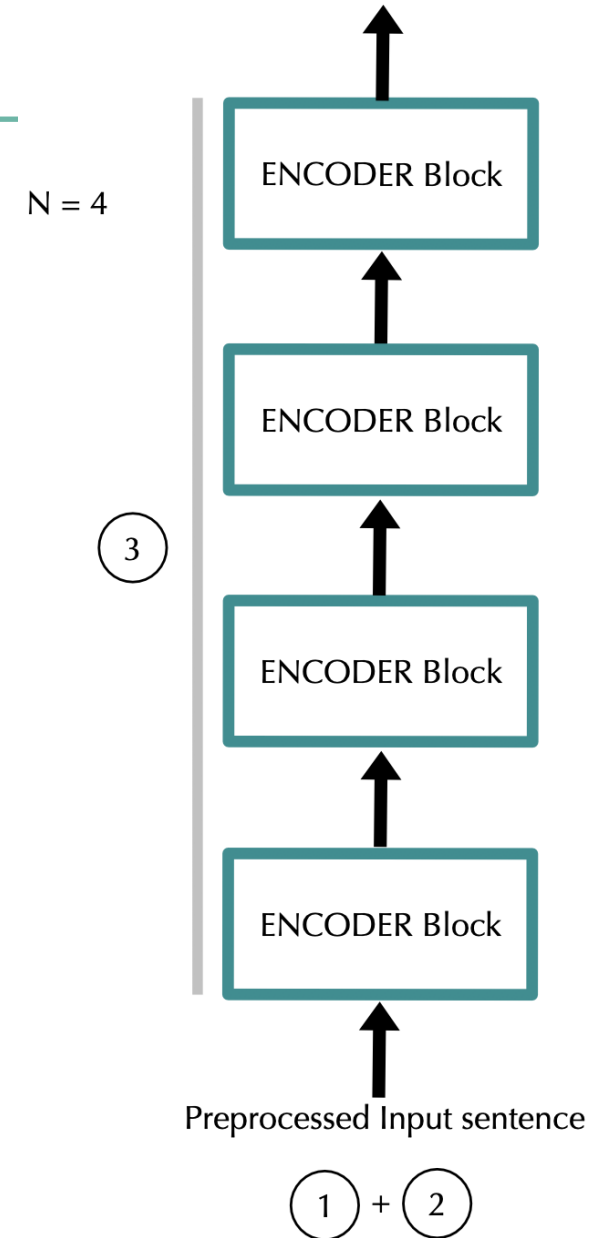
# Information Flow

The data flow through the architecture is as follows:

1. The model represents each token as a vector of `emb_dim` size. We have a matrix of dimensions  $(\text{input\_length}) \times (\text{emb\_dim})$  for a specific input sequence.
2. It then adds positional information (positional encoding). Matrix of dimensions remains the same.
3. The data goes through  $N$  encoder blocks. After this, we obtain the same matrix dimensions

Note:

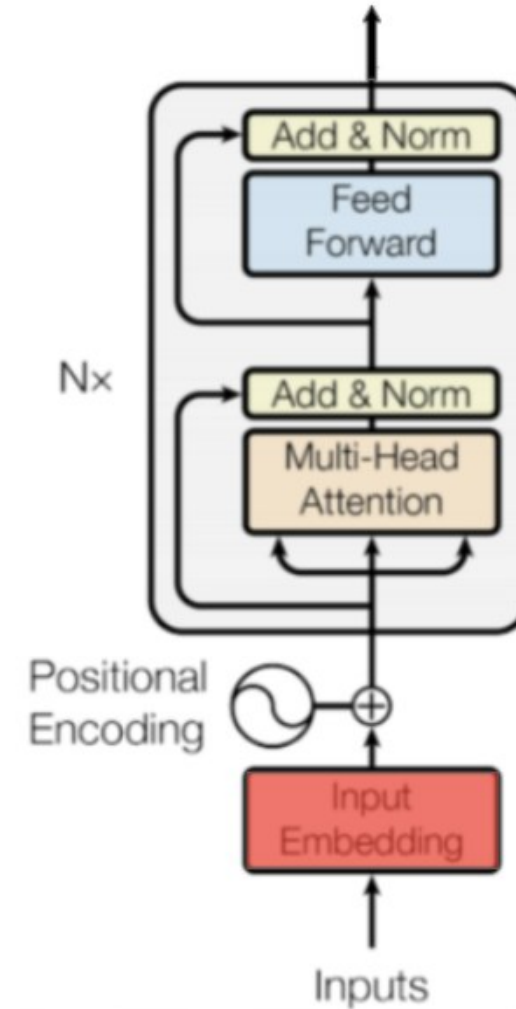
- In the Bert's experiments, they use  $N=12$  or  $24$
- The blocks do not share weights



# From Words to Vectors

From Words to Vectors: Tokenization, numericalization and word embeddings

- Tokenization, numericalization and embeddings do not differ from the way it is done with RNNs.
- Given a sentence in a corpus:
- "Hello, how are you?"
- The first step is to tokenize it:
- "Hello, how are you?"  $\rightarrow$  ["Hello", ",", "how", "are", "you", "?"]



# From Words to Vectors

---

1. This is followed by numericalization, mapping each token to a unique integer in the corpus' vocabulary.
2. ["Hello", ",", ",", "how", "are", "you", "?"]  $\rightarrow$  [34, 90, 15, 684, 55, 193]
3. Next, we get the embedding for each word in the sequence.
4. Each word of the sequence is mapped to an `emb_dim` dimensional vector that the model will learn during training.
5. The elements of those vectors are treated as model parameters and are optimized with back-propagation just like any other weights.

# From Words to Vectors

---

- For each token, we look up the corresponding vector:

$$34 \rightarrow E[34] = [123.4, 0.32, \dots, 94, 32]$$

$$90 \rightarrow E[90] = [83, 34, \dots, 77, 19]$$

$$15 \rightarrow E[15] = [0.2, 50, \dots, 33, 30]$$

$$684 \rightarrow E[684] = [289, 432.98, \dots, 150, 92]$$

$$55 \rightarrow E[55] = [80, 46, \dots, 23, 32]$$

$$193 \rightarrow E[193] = [41, 21, \dots, 74, 33]$$

# From Words to Vectors

---

- Stacking each of the vectors together we obtain a matrix  $Z$  of dimensions (input\_length) x (emb\_dim) :

	$<$	$-$	$d_{emb\_dim}$	$-$	$>$
<i>Hello</i>	123.4	0.32	$\dots$	94	32
<i>,</i>	83	34	$\dots$	77	19
<i>how</i>	0.2	50	$\dots$	33	30
<i>are</i>	289	432.98	$\dots$	150	92
<i>you</i>	80	46	$\dots$	23	32
<i>?</i>	41	21	$\dots$	74	33

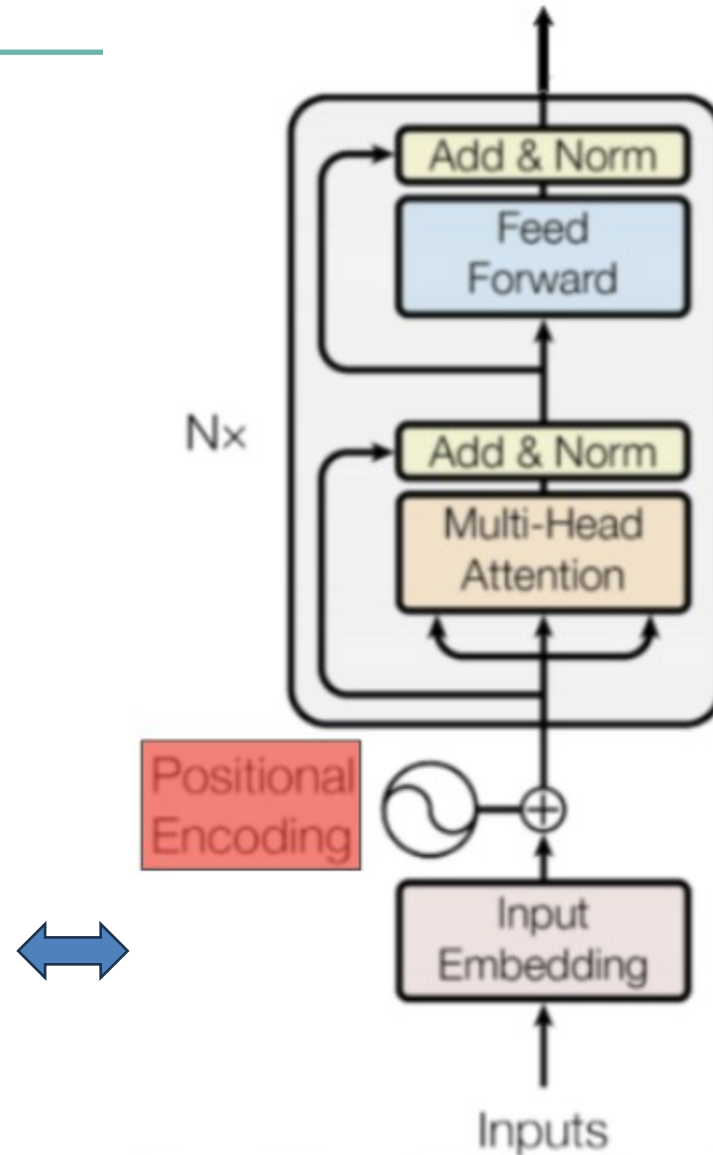
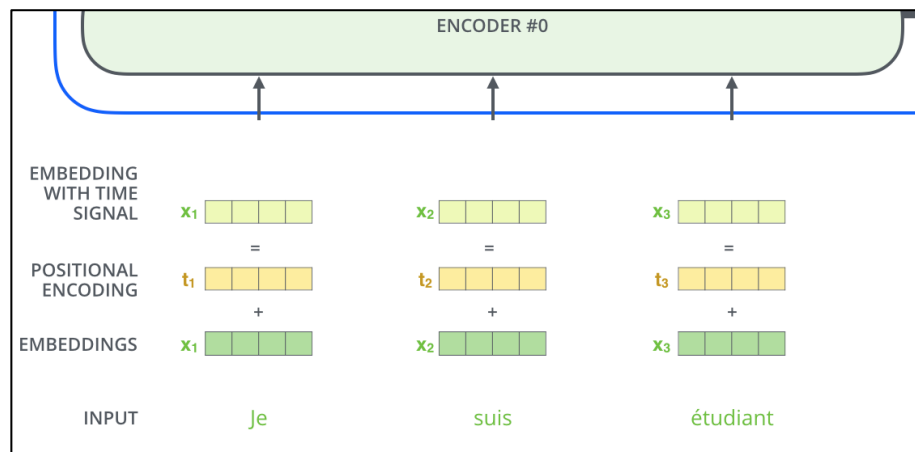
# Padding

---

- It is important to remark that padding was used to make the input sequences in a batch have the same length.
- ["<pad>", "<pad>", "<pad>", "Hello", ",", "how", "are", "you", "?"] →
- [5, 5, 5, 34, 90, 15, 684, 55, 193]

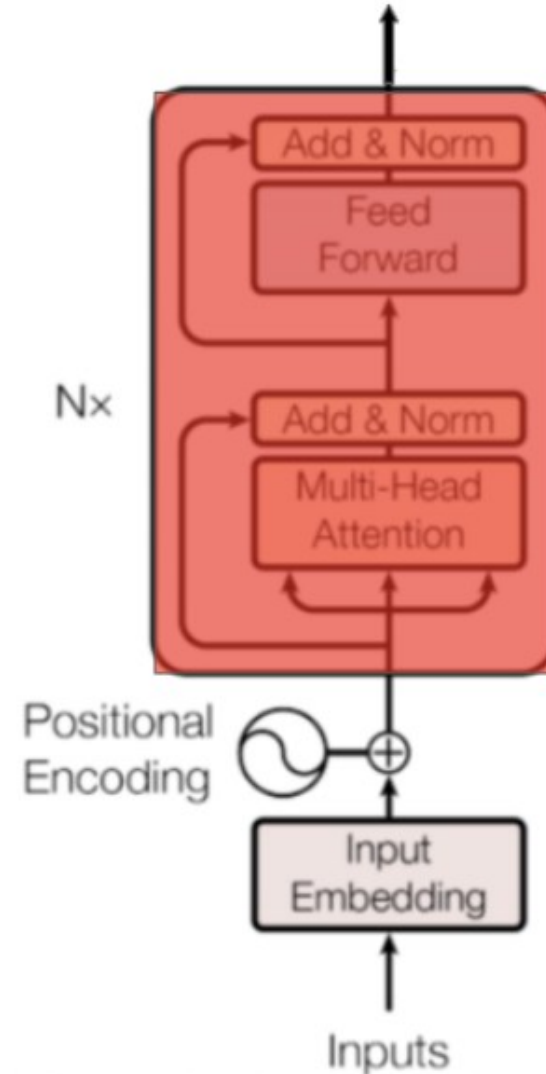
# Positional Encoding

- In BERT the authors used learned positional embeddings.
- At the beginning they add a random numbers



# Encoder block

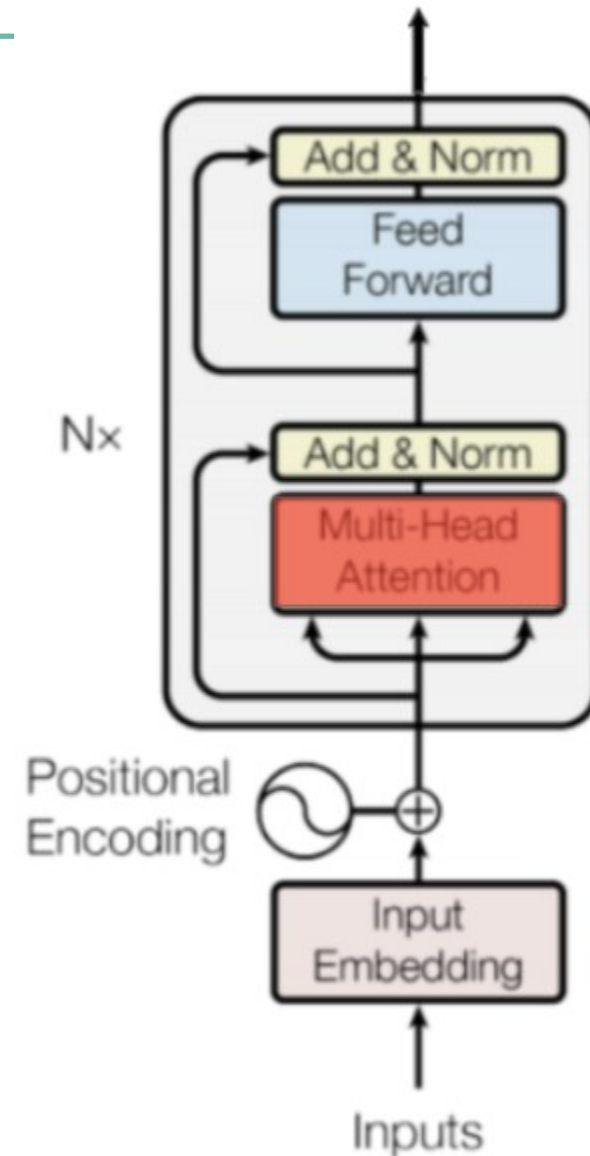
- **A total of N encoder blocks are chained together** to generate the Encoder's output.
- A **specific block** is in charge of **finding relationships** between the input representations and encode them in its output.
- This **iterative process** through the blocks will help the **neural network to capture more complex relationships between words in the input sequence**.





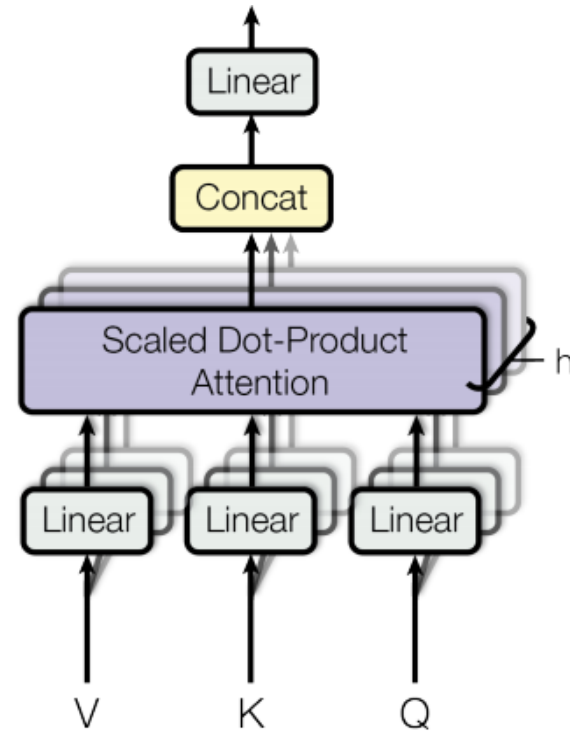
# Multi-Head Attention

- The Transformer uses Multi-Head Attention, which means it computes **attention h different times** with different weight matrices and then concatenates the results together.



# Multi-Head Attention

- The result of each of those parallel computations of attention is called a head.



$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \dots, \text{head}_h)W^O$$

where  $\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$

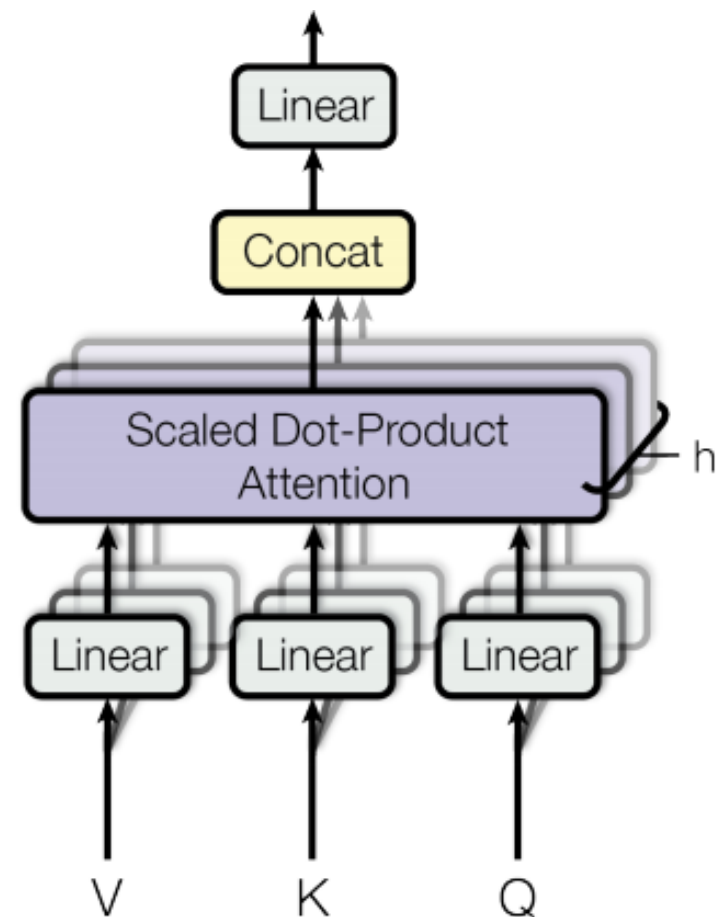
# Multi-Head Attention

- Let's

$$VW_i^V = V_i$$

$$KW_i^K = K_i$$

$$QW_i^Q = Q_i$$



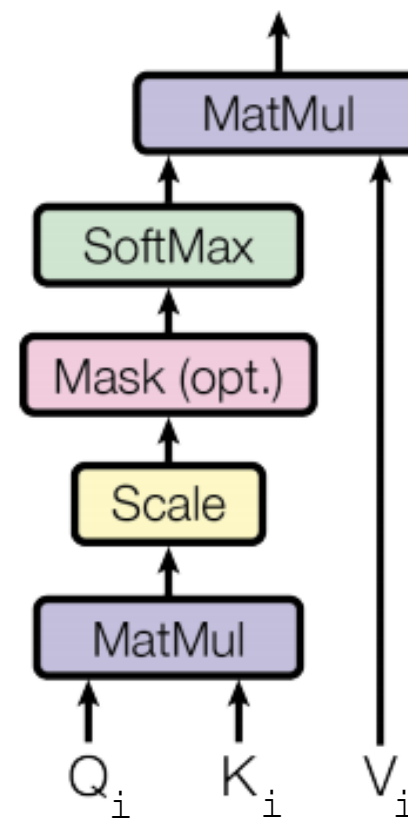
# Scaled Dot-Product Attention

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

- Let's start by looking at the matrix product between  $Q_i$  and  $K_i$  transposed:

$$Q_i K_i^T$$

- Remember  $Q_i$  and  $K_i$  are different projections of the tokens into another dimensional space ( $d_k$ ).
- Therefore, **we can think about the dot product of those projections as a measure of similarity between tokens projections.**

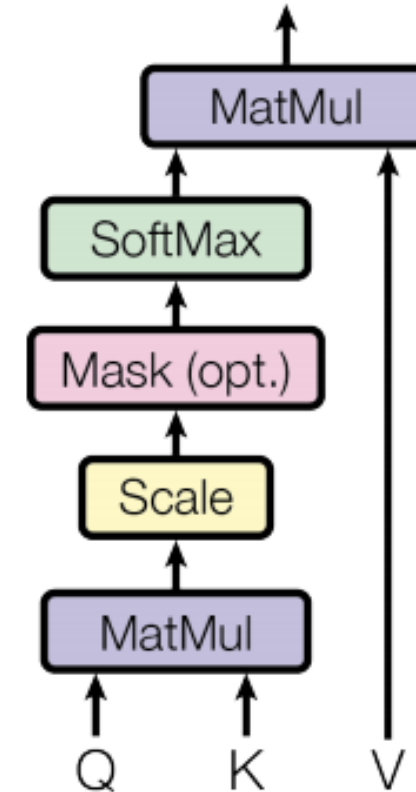


# Scaled Dot-Product Attention

- If we call  $v_i$  and  $u_j$  the projections of the i-th token and the j-th token through  $Q_i$  and  $K_i$  respectively, their dot product can be seen as:

$$v_i u_j = \cos(v_i, u_j) ||v_i||_2 ||u_j||_2$$

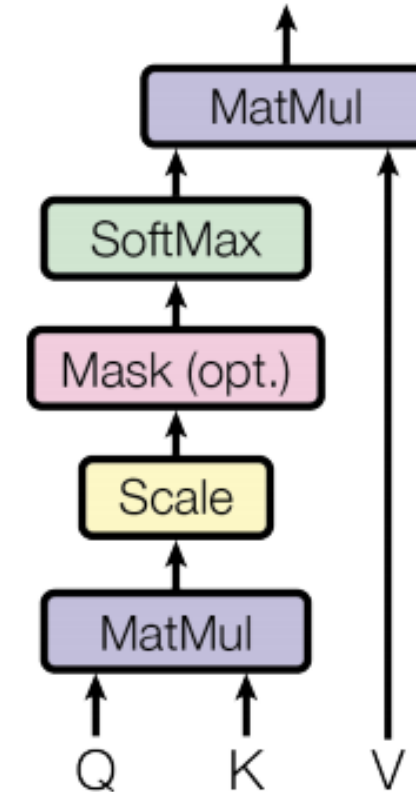
- Thus, this is a **measure of how similar** are the **directions** of  $v_i$  and  $u_j$  and how large are their **lengths** (the closer the direction and the larger the length, the greater the dot product)



# Scaled Dot-Product Attention

- After this multiplication, the **matrix is divided element-wise** by the square root of  $d_k$  for **scaling purposes**.
- **Mask Layer deals with Padding tokens**
- The next step is a **Softmax applied row-wise (one softmax computation for each row)**:

$$\text{Softmax} \left( \frac{Q_i K_i^T}{\sqrt{d_k}} \right)$$



# Softmax (Example)

- Before Softmax

$$\begin{array}{c}
 \text{Hello} \\
 , \\
 \text{how} \\
 \text{are} \\
 \text{you} \\
 ?
 \end{array}
 \begin{pmatrix}
 \text{Hello} & , & \text{how} & \text{are} & \text{you} & ? \\
 78.49 & 43.29 & 1.2 & 41.74 & 91.43 & 74.47 \\
 95.84 & 28.78 & 57.13 & 68.20 & -60.94 & 26.85 \\
 -95.69 & -52.16 & 17.00 & 45.71 & 48.49 & 64.35 \\
 -69.92 & 85.16 & 94.94 & 91.04 & -92.83 & 77.49 \\
 65.85 & 55.85 & 62.54 & -97.46 & 76.38 & 13.20 \\
 -30.05 & -4.52 & 76.02 & 42.35 & 15.29 & 63.61
 \end{pmatrix}
 \Rightarrow$$

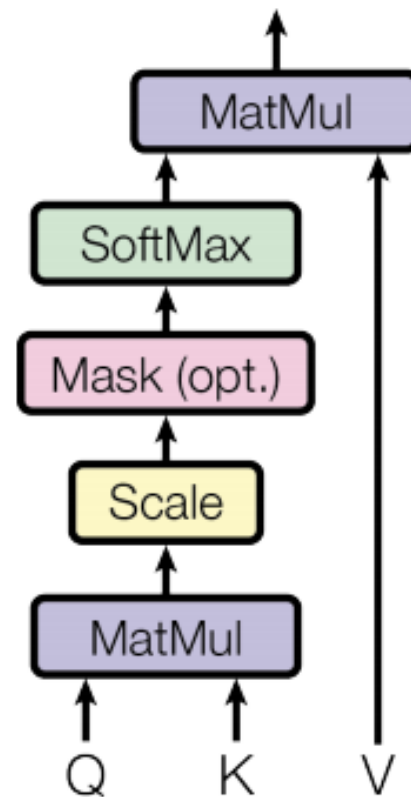
- After Softmax

$$\begin{array}{c}
 \text{Hello} \\
 , \\
 \text{how} \\
 \text{are} \\
 \text{you} \\
 ?
 \end{array}
 \begin{pmatrix}
 \text{Hello} & , & \text{how} & \text{are} & \text{you} & ? \\
 72.40 * 10^{-06} & 1.23 * 10^{-21} & 6.51 * 10^{-40} & 2.62 * 10^{-22} & 9.99 * 10^{-01} & 4.30 * 10^{-08} \\
 1.00 * 10^{+00} & 7.51 * 10^{-30} & 1.54 * 10^{-17} & 9.91 * 10^{-13} & 8.15 * 10^{-69} & 1.09 * 10^{-30} \\
 3.12 * 10^{-70} & 2.51 * 10^{-51} & 2.72 * 10^{-21} & 8.03 * 10^{-09} & 1.29 * 10^{-07} & 9.99 * 10^{-01} \\
 2.47 * 10^{-72} & 5.54 * 10^{-05} & 9.80 * 10^{-01} & 1.98 * 10^{-02} & 2.77 * 10^{-82} & 2.58 * 10^{-08} \\
 2.67 * 10^{-05} & 1.21 * 10^{-09} & 9.75 * 10^{-07} & 3.17 * 10^{-76} & 9.99 * 10^{-01} & 3.64 * 10^{-28} \\
 8.59 * 10^{-47} & 1.05 * 10^{-35} & 9.99 * 10^{-01} & 2.38 * 10^{-15} & 4.21 * 10^{-27} & 4.07 * 10^{-06}
 \end{pmatrix}
 \begin{array}{l}
 = 1 \\
 = 1 \\
 = 1 \\
 = 1 \\
 = 1 \\
 = 1
 \end{array}$$

# Scaled Dot-Product Attention

- The result would be rows with numbers between zero and one that sum to one. Finally, the result is multiplied by  $V_i$  to get the result of the head.

$$\text{Softmax} \left( \frac{Q_i K_i^T}{\sqrt{d_k}} \right) V_i$$





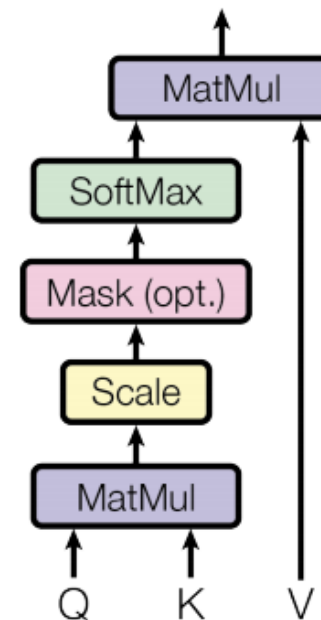
# Example #1

- Let's propose a dummy example. Suppose that the resulting first row of:

$$\text{Softmax} \left( \frac{Q_i K_i^T}{\sqrt{d_k}} \right)$$

- is [0,0,0,0,1,0]. So:

$$\begin{array}{c}
 \text{Hello} \\
 , \\
 \text{how} \\
 \text{are} \\
 \text{you} \\
 ?
 \end{array}
 \begin{pmatrix}
 \text{Hello} & , & \text{how} & \text{are} & \text{you} & ? \\
 0 & 0 & 0 & 0 & 1 & 0 \\
 \dots & \dots & \dots & \dots & \dots & \dots \\
 \dots & \dots & \dots & \dots & \dots & \dots \\
 \dots & \dots & \dots & \dots & \dots & \dots \\
 \dots & \dots & \dots & \dots & \dots & \dots
 \end{pmatrix}
 \begin{pmatrix}
 d_v \\
 v_{\text{Hello}} \\
 v_{\text{,}} \\
 v_{\text{how}} \\
 v_{\text{are}} \\
 v_{\text{you}} \\
 v_{\text{?}}
 \end{pmatrix}
 =
 \begin{array}{c}
 \text{Hello} \\
 , \\
 \text{how} \\
 \text{are} \\
 \text{you} \\
 ?
 \end{array}
 \begin{pmatrix}
 d_v \\
 v_{\text{you}} \\
 \dots \\
 \dots \\
 \dots \\
 \dots \\
 \dots
 \end{pmatrix}$$



# Example #1

---

$$\begin{array}{c}
 \text{Hello} \\
 , \\
 \text{how} \\
 \text{are} \\
 \text{you} \\
 ?
 \end{array}
 \begin{array}{c}
 \text{Hello} \\
 , \\
 \text{how} \\
 \text{are} \\
 \text{you} \\
 ?
 \end{array}
 \begin{pmatrix}
 0 & 0 & 0 & 0 & 1 & 0 \\
 \dots & \dots & \dots & \dots & \dots & \dots \\
 \dots & \dots & \dots & \dots & \dots & \dots \\
 \dots & \dots & \dots & \dots & \dots & \dots \\
 \dots & \dots & \dots & \dots & \dots & \dots \\
 \dots & \dots & \dots & \dots & \dots & \dots
 \end{pmatrix}
 \begin{array}{c}
 d_v \\
 v_{\text{Hello}} \\
 v_{,} \\
 v_{\text{how}} \\
 v_{\text{are}} \\
 v_{\text{you}} \\
 v_{?}
 \end{array}
 =
 \begin{array}{c}
 \text{Hello} \\
 , \\
 \text{how} \\
 \text{are} \\
 \text{you} \\
 ?
 \end{array}
 \begin{array}{c}
 d_v \\
 v_{\text{you}} \\
 \dots \\
 \dots \\
 \dots \\
 \dots \\
 \dots
 \end{array}$$

- Observe that in this case the **word “hello” ends up with a representation** based on the 5th token **“you” of the input for that head.**
- Supposing an equivalent example for the rest of the heads. **The word “Hello” will be now represented by the concatenation of the different projections of other words.**
- **The network will learn over training time which relationships are more useful and will relate tokens to each other based on these relationships.**

## Example #2

- Let us now complicate the example a little bit more.

$$\begin{matrix} & \textit{Hello} & , & \textit{how} & \textit{are} & \textit{you} & ? & d_v \\ \textit{Hello} & \left( \begin{matrix} 0.1 & 0 & 0.06 & 0.1 & 0.6 & 0.14 \\ \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots \end{matrix} \right) & \begin{pmatrix} v_{\textit{Hello}} \\ v, \\ v_{\textit{how}} \\ v_{\textit{are}} \\ v_{\textit{you}} \\ v? \end{pmatrix} & = \end{matrix}$$

- This results in

$$Hello \left( \begin{array}{c} < - - - - - d_v - - - - - > \\ 0.1v_{Hello} + 0v, + 0.06v_{how} + 0.1v_{are} + 0.6v_{you} + 0.14v? \\ ..... \\ ..... \\ ..... \\ ..... \\ ..... \end{array} \right)$$

## Example #2

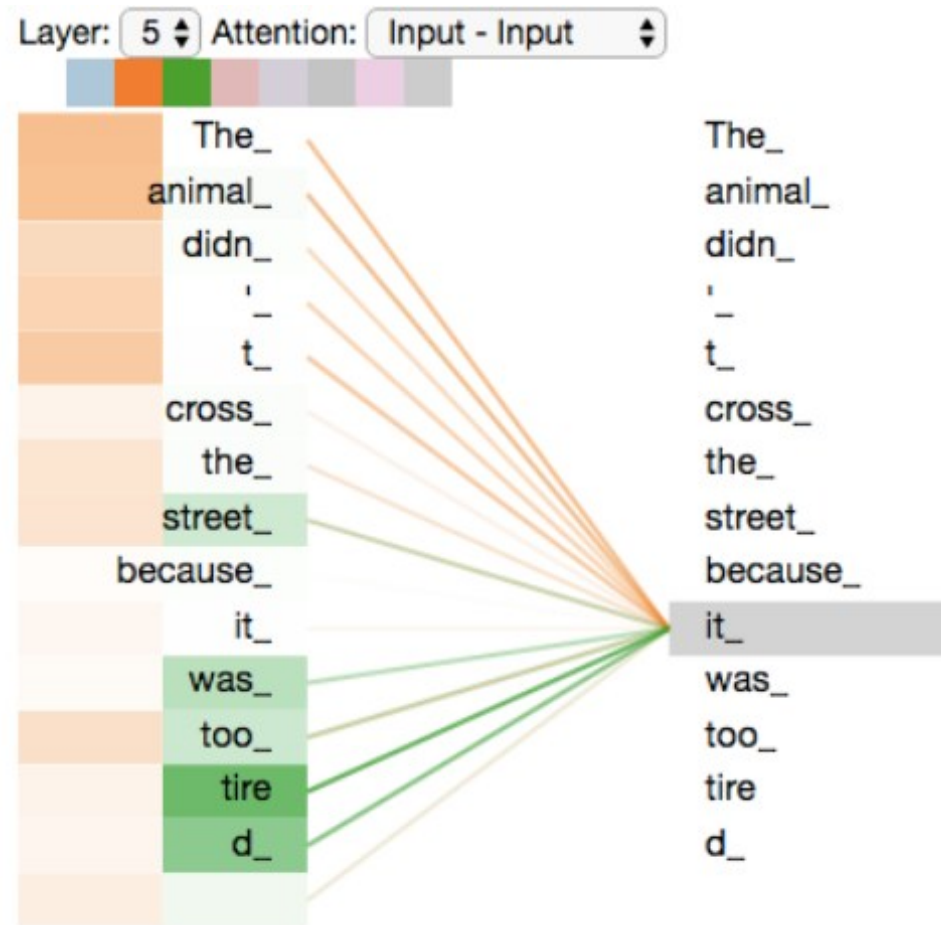
---

$$\begin{matrix} \text{Hello} \\ , \\ \text{how} \\ \text{are} \\ \text{you} \\ ? \end{matrix} \left( \begin{array}{c} \langle \text{-----} d_v \text{-----} \rangle \\ 0.1v_{\text{Hello}} + 0v_{,} + 0.06v_{\text{how}} + 0.1v_{\text{are}} + 0.6v_{\text{you}} + 0.14v_{?} \\ \dots\dots\dots \\ \dots\dots\dots \\ \dots\dots\dots \\ \dots\dots\dots \\ \dots\dots\dots \end{array} \right)$$

- Observe that we can think about the resulting **representation of “Hello”** as **a weighted combination (centroid) of the projected vectors through  $V_i$  of the input tokens.**

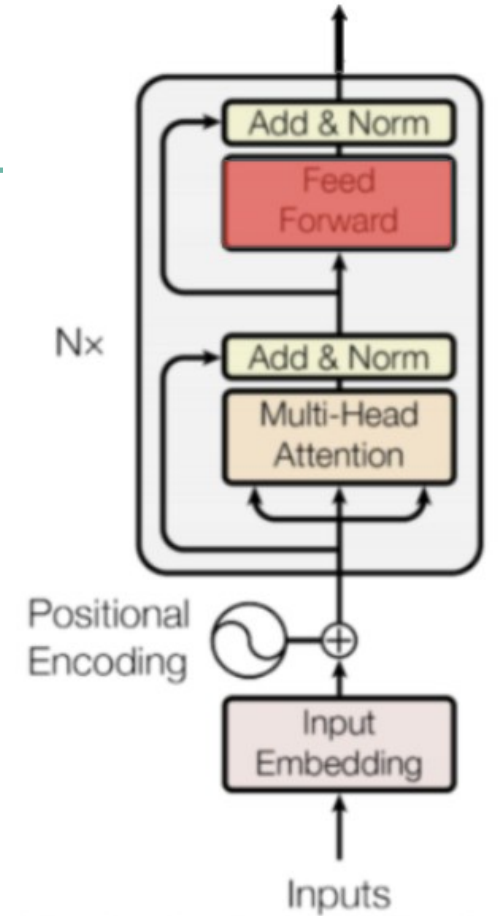
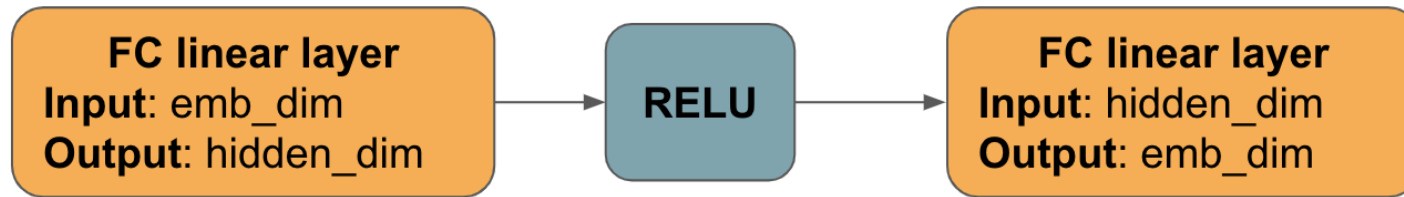
## Example #3

---



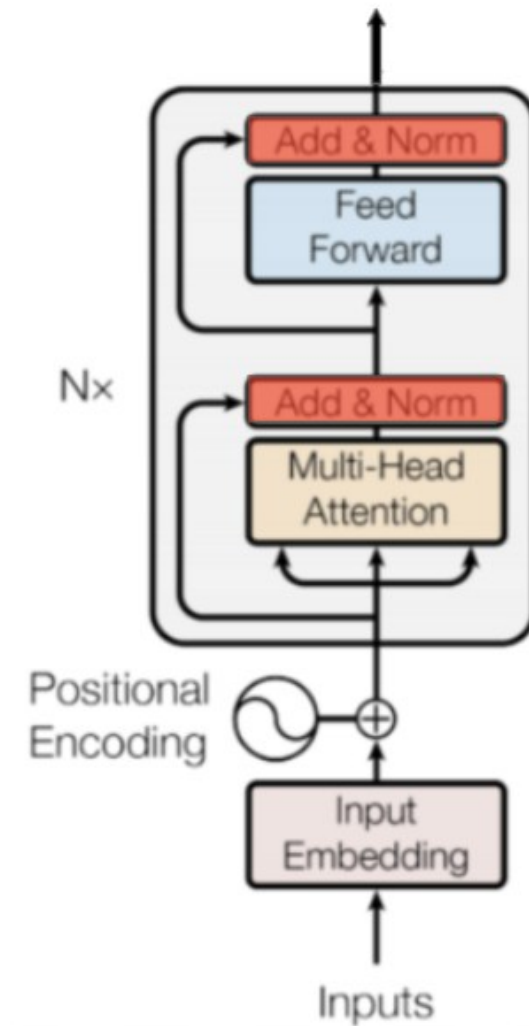
# Position-wise Feed-Forward Network

- This step is composed of the following layers:



# Dropout, Add & Norm

- In this layer is applied a dropout at 10% in the input.
- This result is added to the original input.
- It is like telling the token:
- **“Learn the relationship with the rest of the tokens, but don’t forget what we already learned about yourself!”**
- A token-wise/row-wise normalization is computed with the mean and standard deviation of each row (improves the stability of the network).
- The output of these layers is:  $LayerNorm(x + Dropout(Sublayer(x)))$



# BERT'S TRAINING

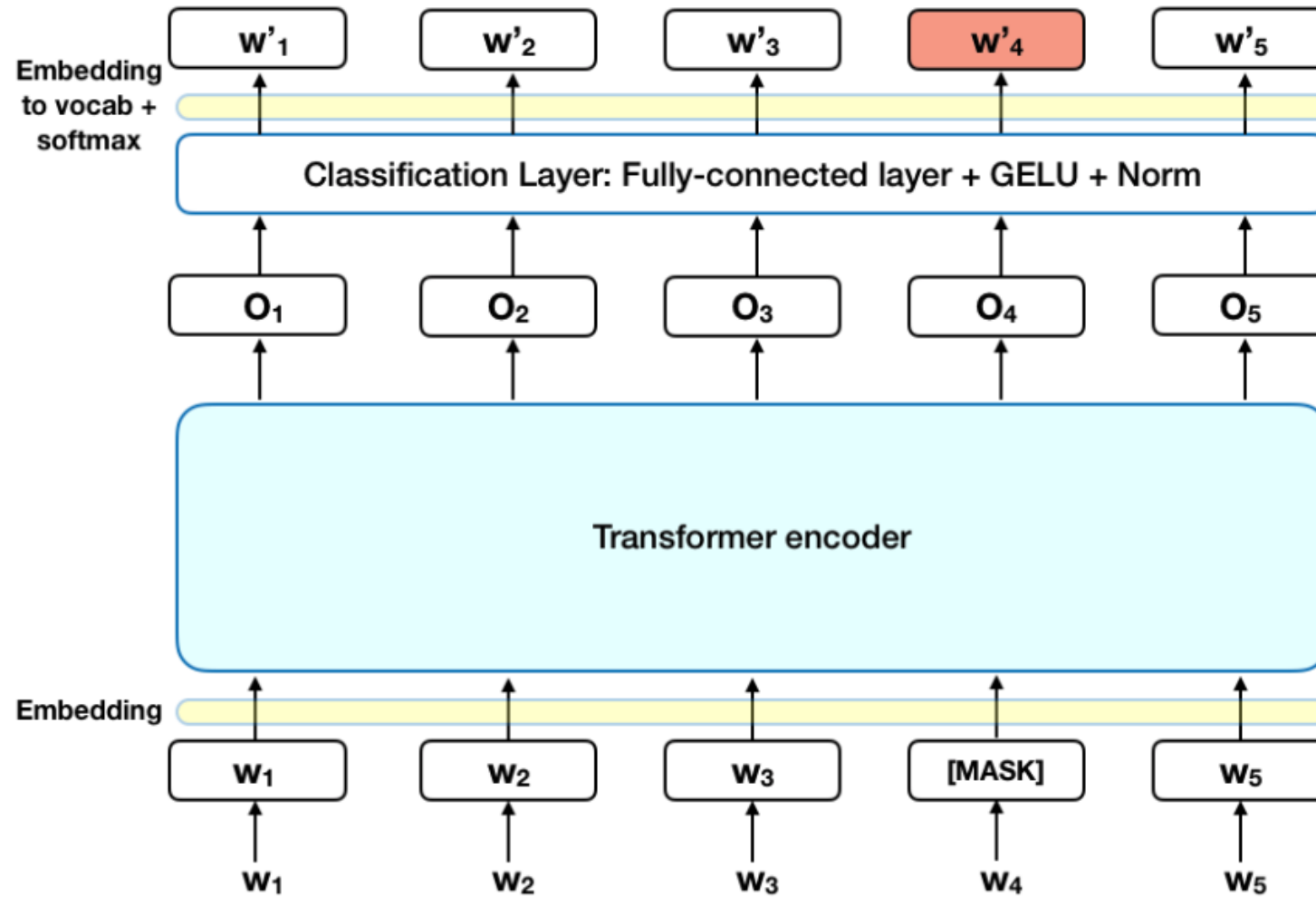


# BERT'S TRAINING

---

- When training language models, there is a challenge of defining a prediction goal.
- Many models predict the next word in a sequence (e.g. "The child came home from \_\_\_\_"), a directional approach which inherently limits context learning.
- To overcome this challenge, BERT uses two training strategies simultaneously: Masked Language Model and Next Sentence Prediction.

# Task1: Masked Language Model



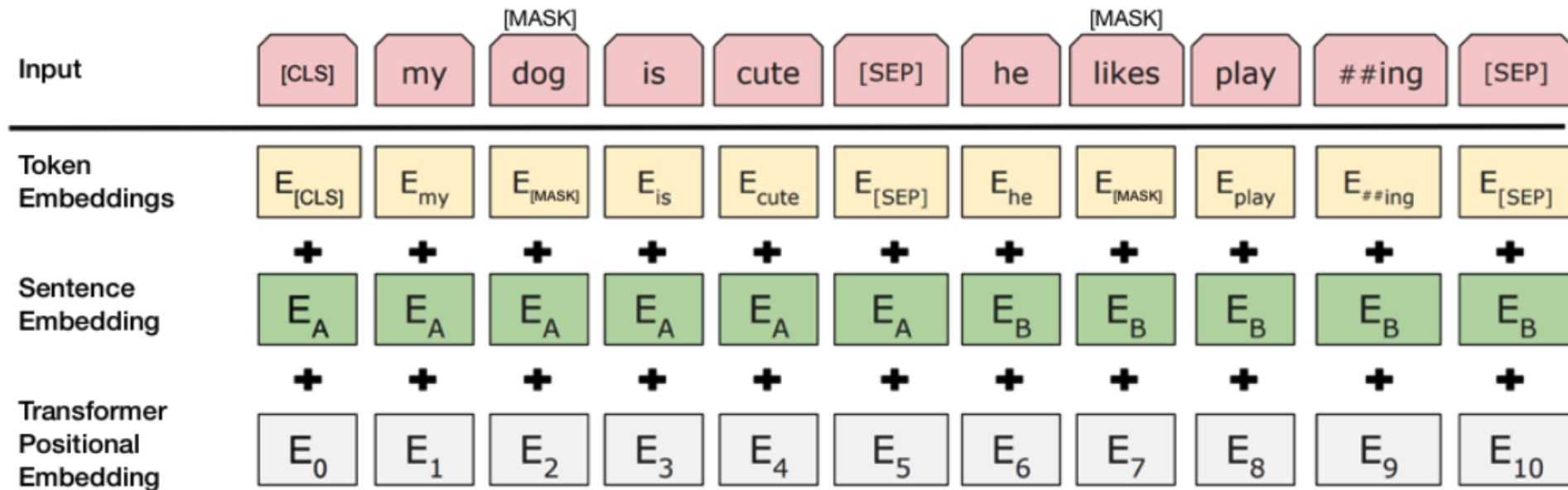
# Task1: Masked Language Model

---

- The **Masked Language Model asks the model to predict**, not the next word for a sequence of words, but rather **random words from within the sequence**.
- In technical terms, the prediction of the output words requires:
  - Adding a **classification layer** on top of the encoder output.
  - **Multiplying the output vectors by the embedding matrix**, transforming them into the vocabulary dimension.
  - **Calculating the probability of each word in the vocabulary with softmax**.
- Note: Before feeding word sequences into BERT, 15% of the words in each sequence are replaced with a [MASK] token.

## Task 2: Next Sentence Prediction

---



## Task 2: Next Sentence Prediction

---

- This task consists of giving the model two sentences and asking it to predict if the second sentence follows the first in a corpus or not.
- To help the model distinguish between the two sentences in training, the input is processed in the following way before entering the model:
  - A [CLS] token is inserted at the beginning of the first sentence and a [SEP] token is inserted at the end of each sentence.
  - A sentence embedding indicating Sentence A or Sentence B is added to each token.
  - A positional embedding is added to each token to indicate its position in the sequence.
- During training, 50% of the inputs are a pair in which the second sentence is the subsequent sentence in the original document, while in the other 50% a random sentence from the corpus is chosen as the second sentence.

## Task 2: Next Sentence Prediction

---

- To predict if the second sentence is indeed connected to the first, the following steps are performed:
  - The entire input sequence goes through the Transformer model.
  - The output of the [CLS] token is transformed into a  $2 \times 1$  shaped vector, using a simple classification layer (learned matrices of weights and biases).
  - Calculating the probability of IsNextSequence with softmax.

Fine-tuning

# Fine-tuning

---

- **BERT** can be used for a **wide variety of language tasks**, while only adding a small layer to the core model:
- 1. **Classification tasks** such as sentiment analysis are done similarly to **Next Sentence classification**, by adding a classification layer on top of the Transformer output for the [CLS] token.
- 2. In **Question Answering tasks** (e.g. SQuAD v2.0), the software receives a question regarding a text sequence and is required to mark the answer in the sequence. Using BERT, a Q&A model can be trained by learning **two extra vectors that mark the beginning and the end of the answer**.
- 3. In **Named Entity Recognition** (NER), the software receives a text sequence and is required to mark the various types of entities (Person, Organization, Date, etc) that appear in the text. Using BERT, a NER model **can be trained by feeding the output vector of each token into a classification layer that predicts the NER label**.



# References

---

BERT Explained: State of the art language model for NLP

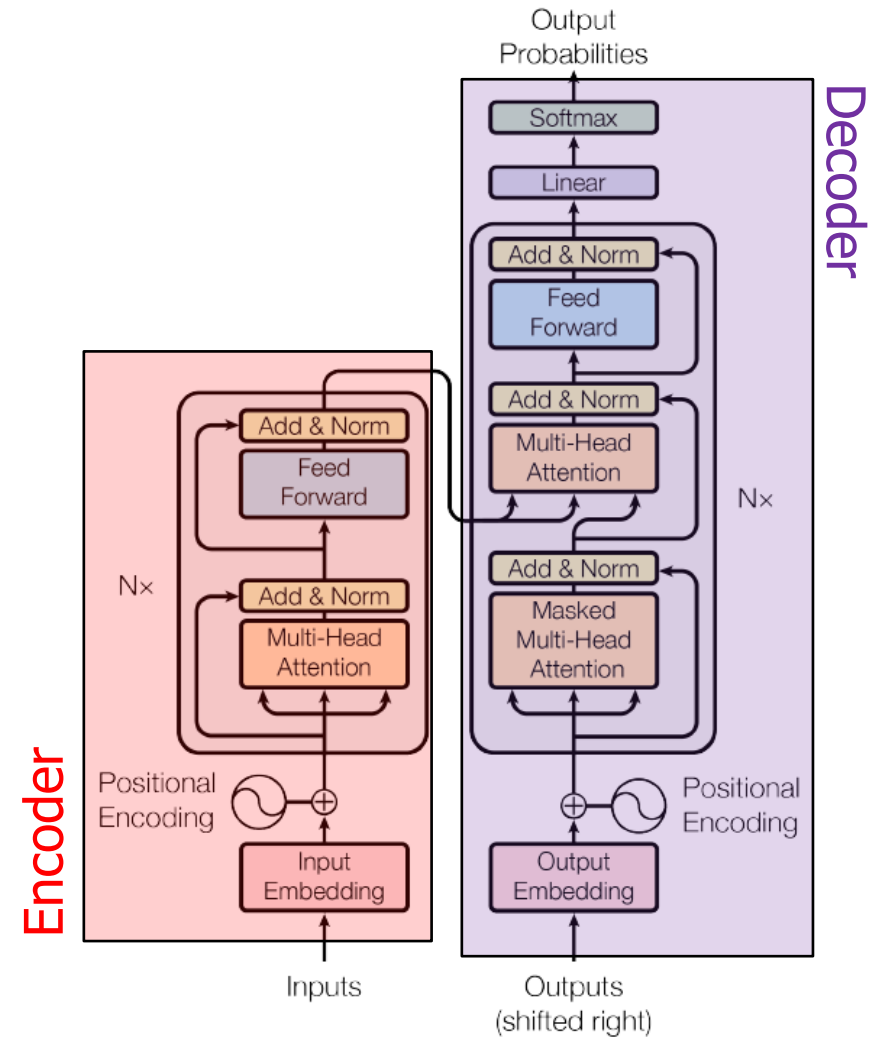
Dissecting BERT Part 1: The Encoder

Understanding BERT Part 2: BERT Specifics

# GPT and LLMs

# Transformer (Google 2017)

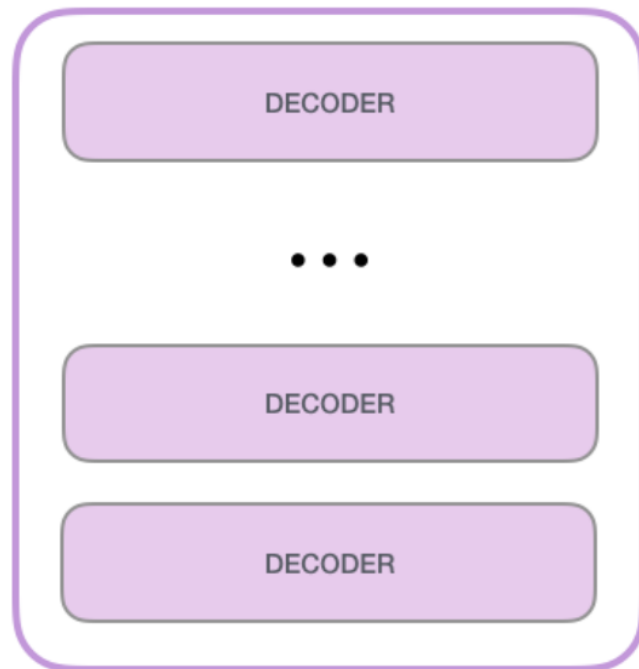
- Transformer was developed for solving text translation problem.
- The Transformer architecture consists of an Encoder and a Decoder and the Encoder's output is an input to the Decoder.



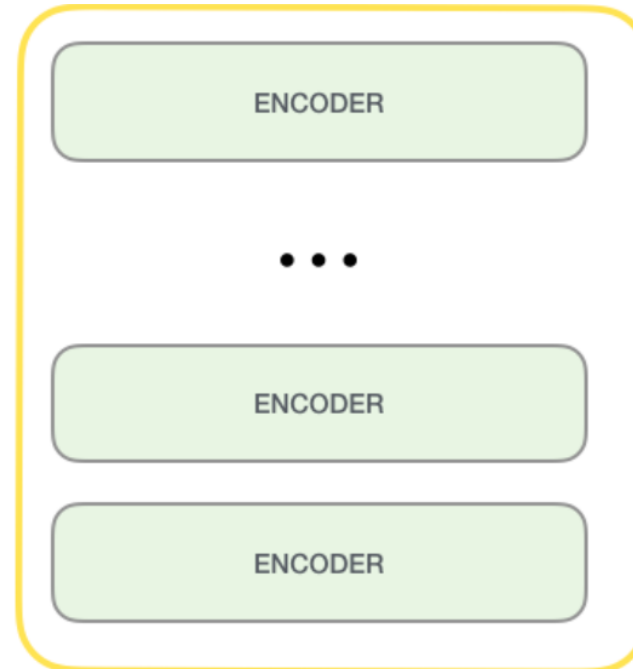
# GPT-3 VS BERT

---

## GPT-3



## BERT

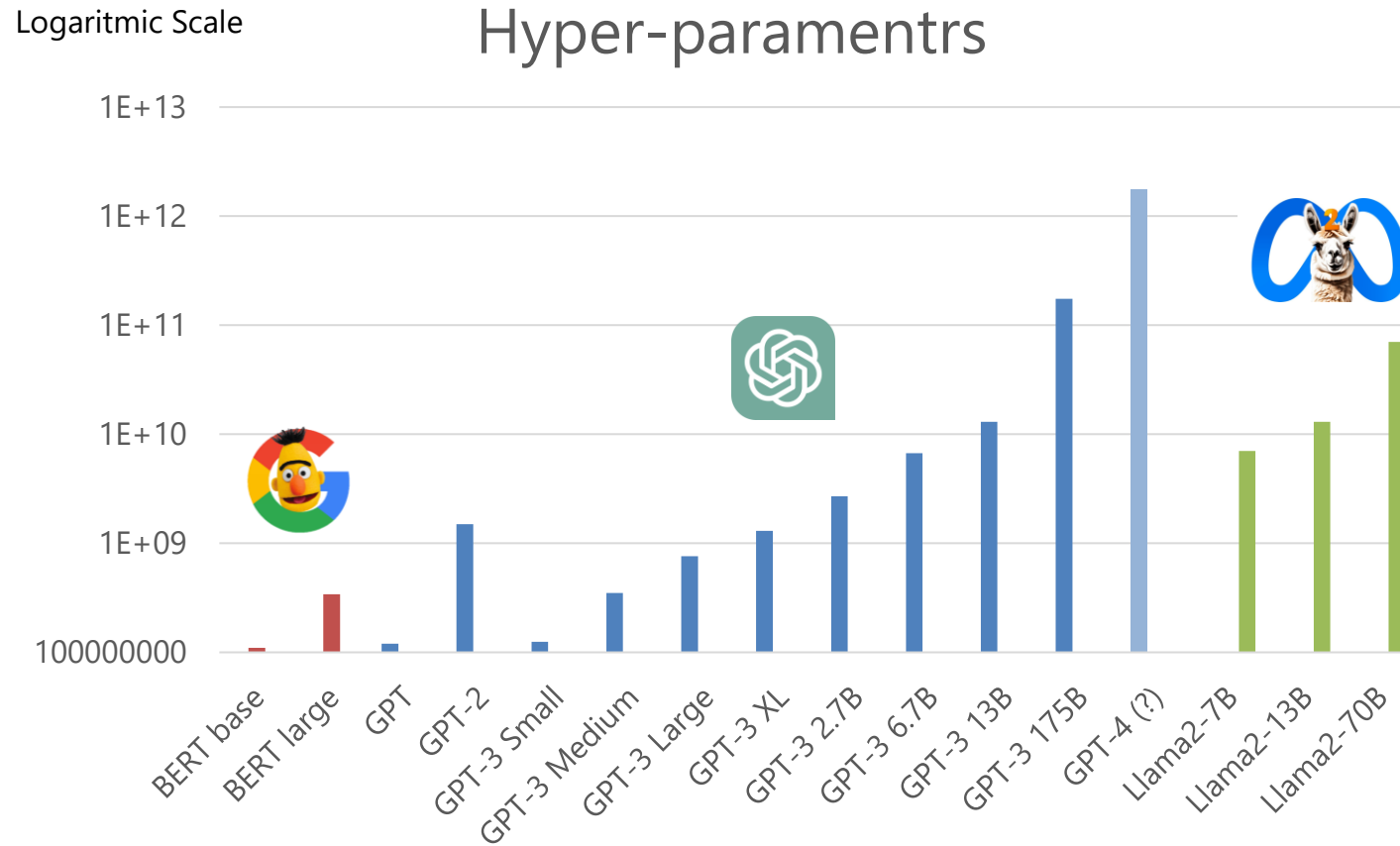


# Hyper-paramentrs

---

Model Name	$n_{\text{params}}$	$n_{\text{layers}}$	$d_{\text{model}}$	$n_{\text{heads}}$	$d_{\text{head}}$	Batch Size	Learning Rate
GPT-3 Small	125M	12	768	12	64	0.5M	$6.0 \times 10^{-4}$
GPT-3 Medium	350M	24	1024	16	64	0.5M	$3.0 \times 10^{-4}$
GPT-3 Large	760M	24	1536	16	96	0.5M	$2.5 \times 10^{-4}$
GPT-3 XL	1.3B	24	2048	24	128	1M	$2.0 \times 10^{-4}$
GPT-3 2.7B	2.7B	32	2560	32	80	1M	$1.6 \times 10^{-4}$
GPT-3 6.7B	6.7B	32	4096	32	128	2M	$1.2 \times 10^{-4}$
GPT-3 13B	13.0B	40	5140	40	128	2M	$1.0 \times 10^{-4}$
GPT-3 175B or “GPT-3”	175.0B	96	12288	96	128	3.2M	$0.6 \times 10^{-4}$

# Hyper-paramentrs



## Dataset used to Train

---

Dataset	Quantity (tokens)	Weight in training mix	Epochs elapsed when training for 300B tokens
Common Crawl (filtered)	410 billion	60%	0.44
WebText2	19 billion	22%	2.9
Books1	12 billion	8%	1.9
Books2	55 billion	8%	0.43
Wikipedia	3 billion	3%	3.4

# GPT Family

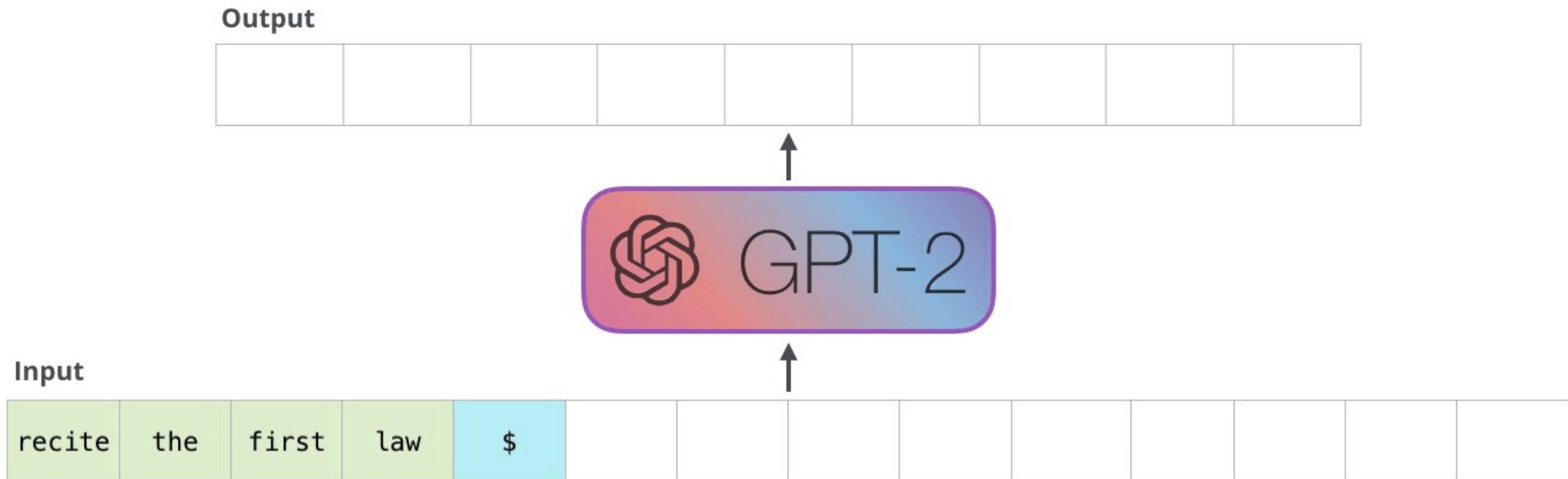
---





# Generative Model

# GPT-2 – Difference from BERT

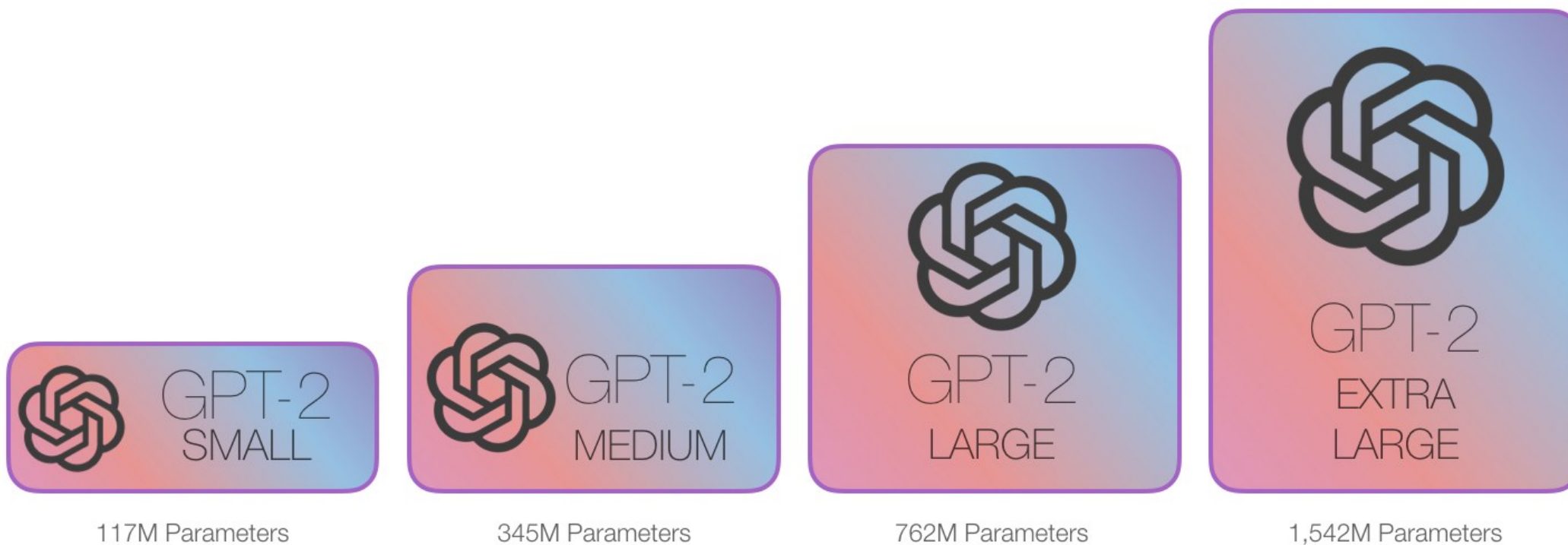


# GPT-2

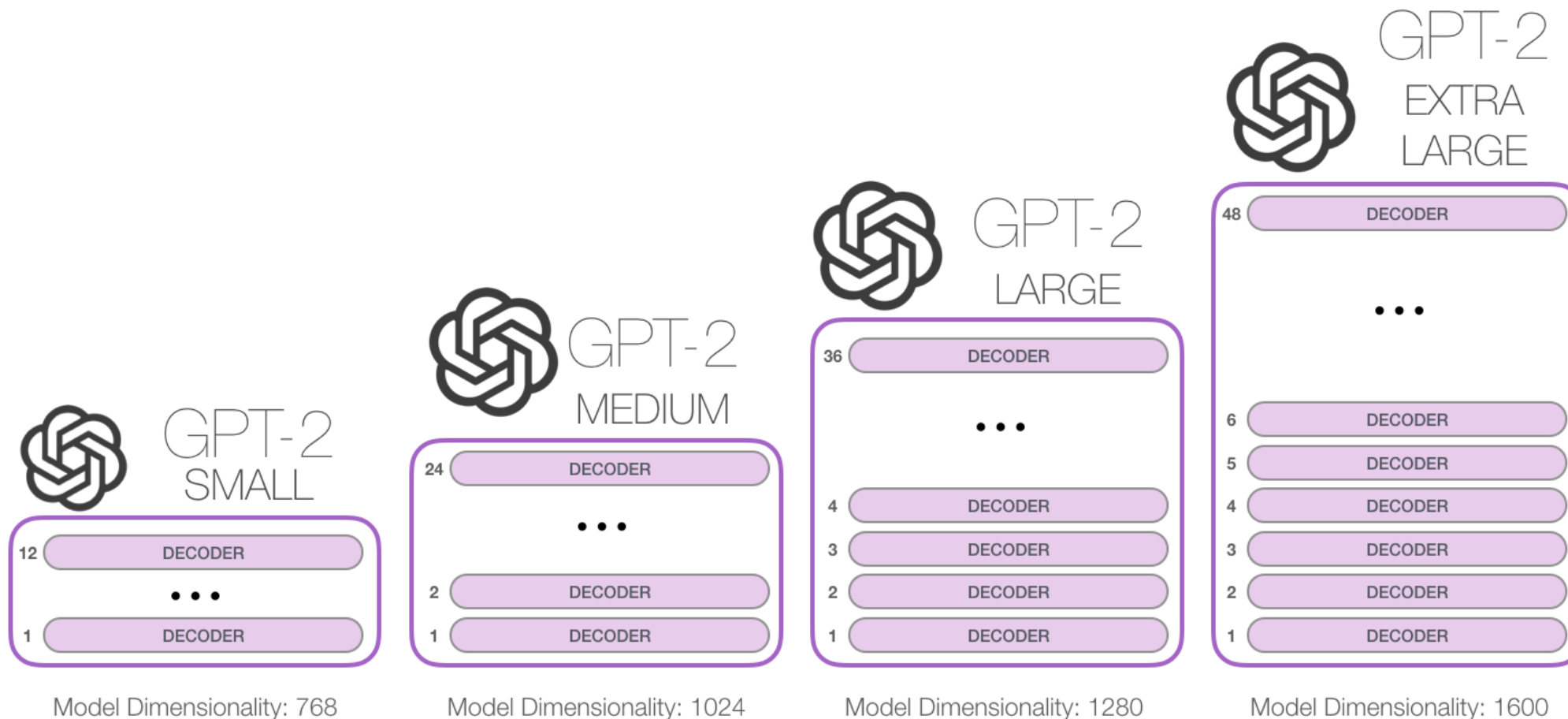
**Output**



# GPT-2



# GPT-2

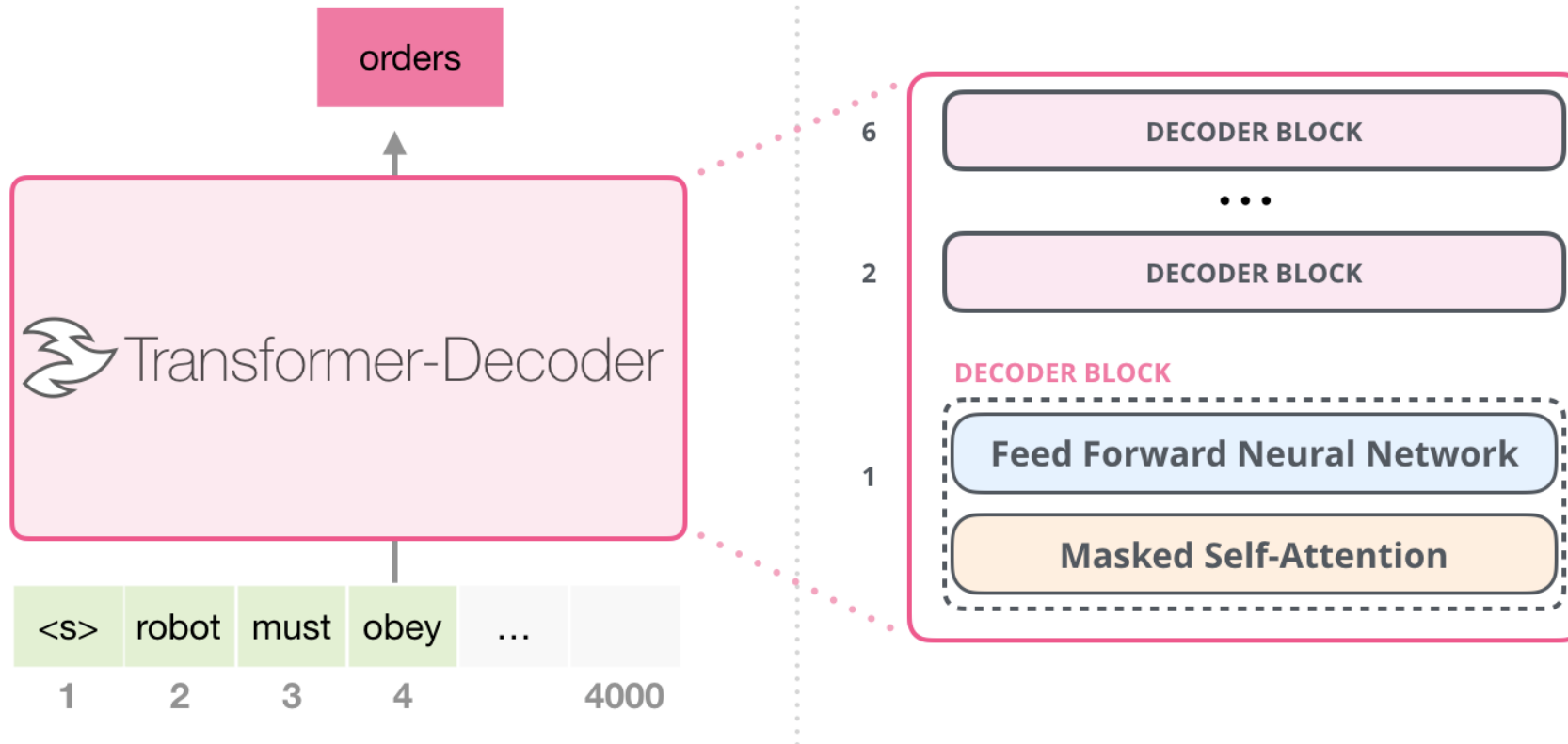


# GPT2 – Key Concept

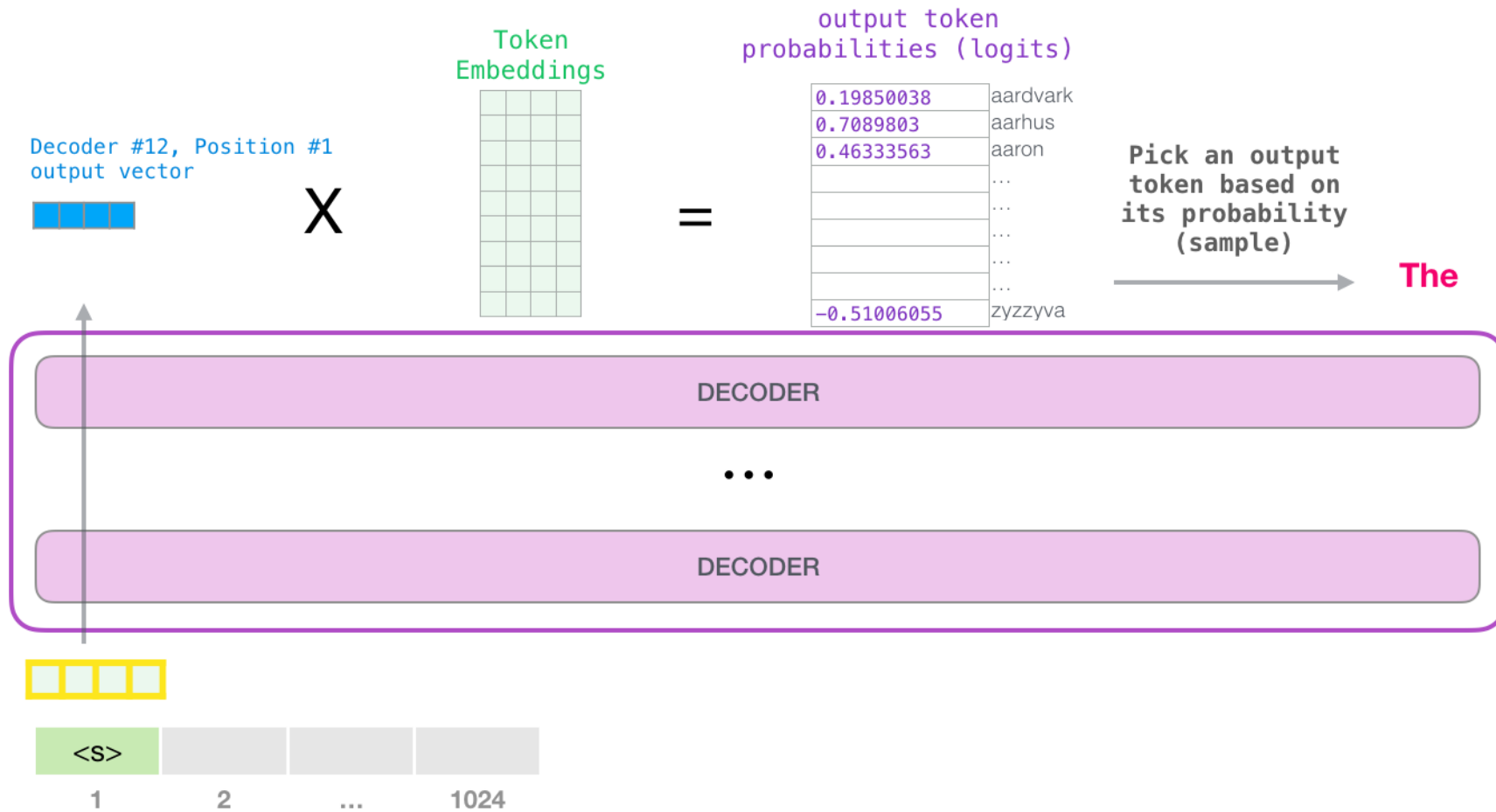
## Input Text Sequence:

robot	must	obey	orders
1	2	3	4

## GPT2:

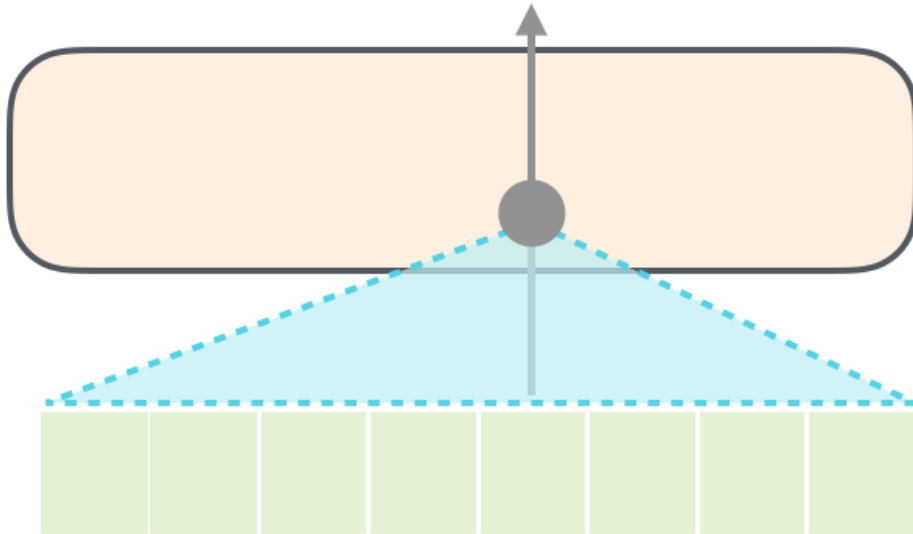


# GPT2 – Key Concept

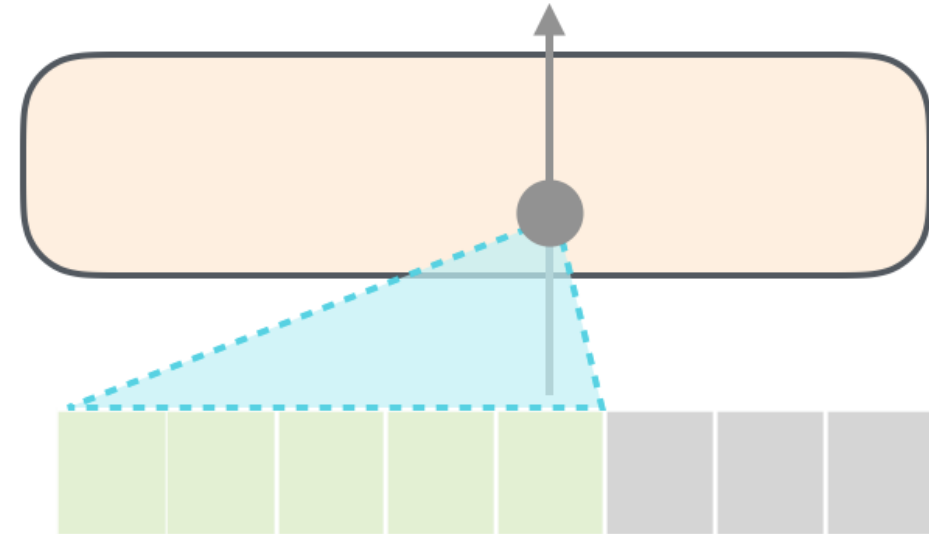


# Masked Self-Attention

## Self-Attention

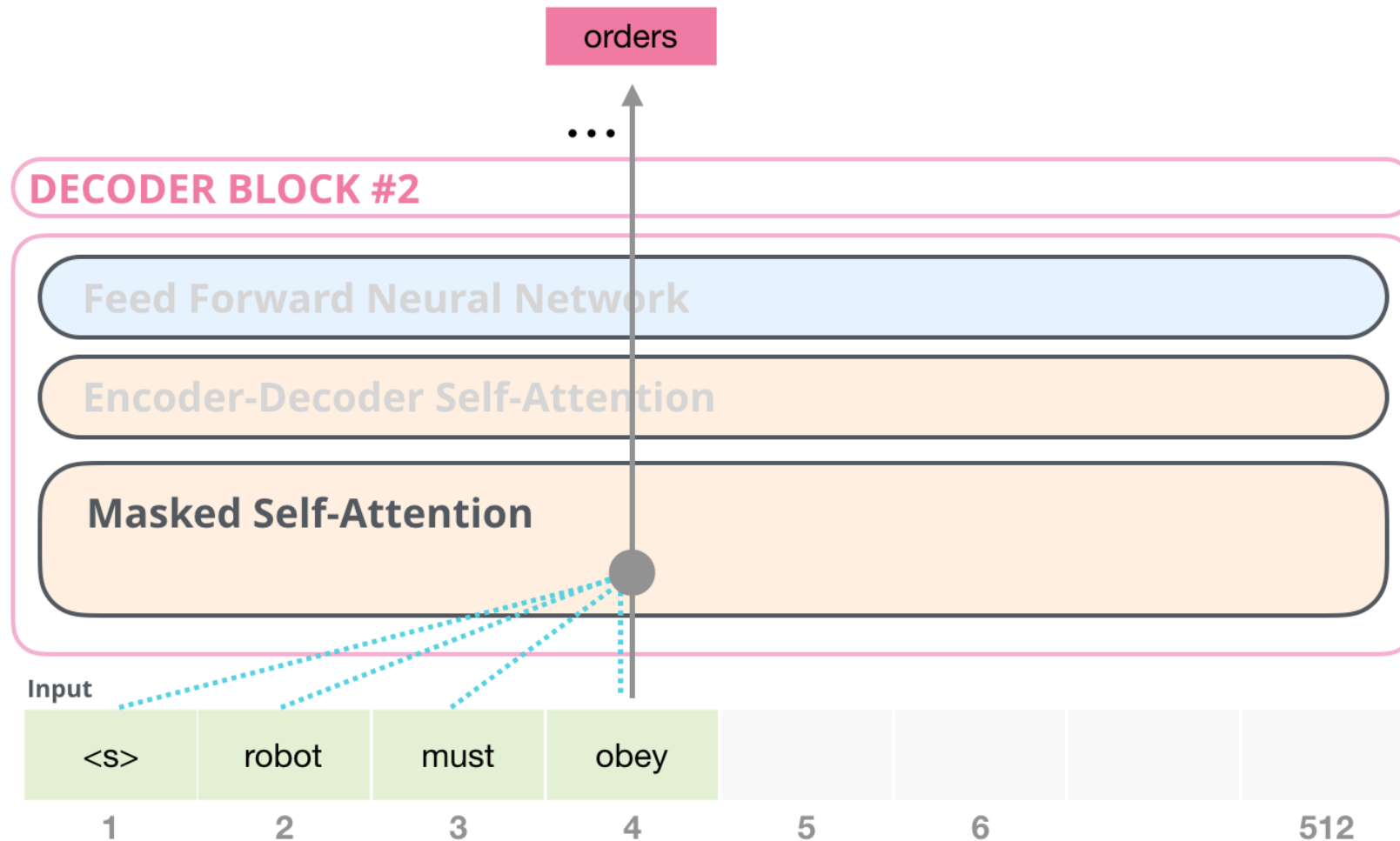


## Masked Self-Attention





# Masked Self-Attention



# Machine Translation

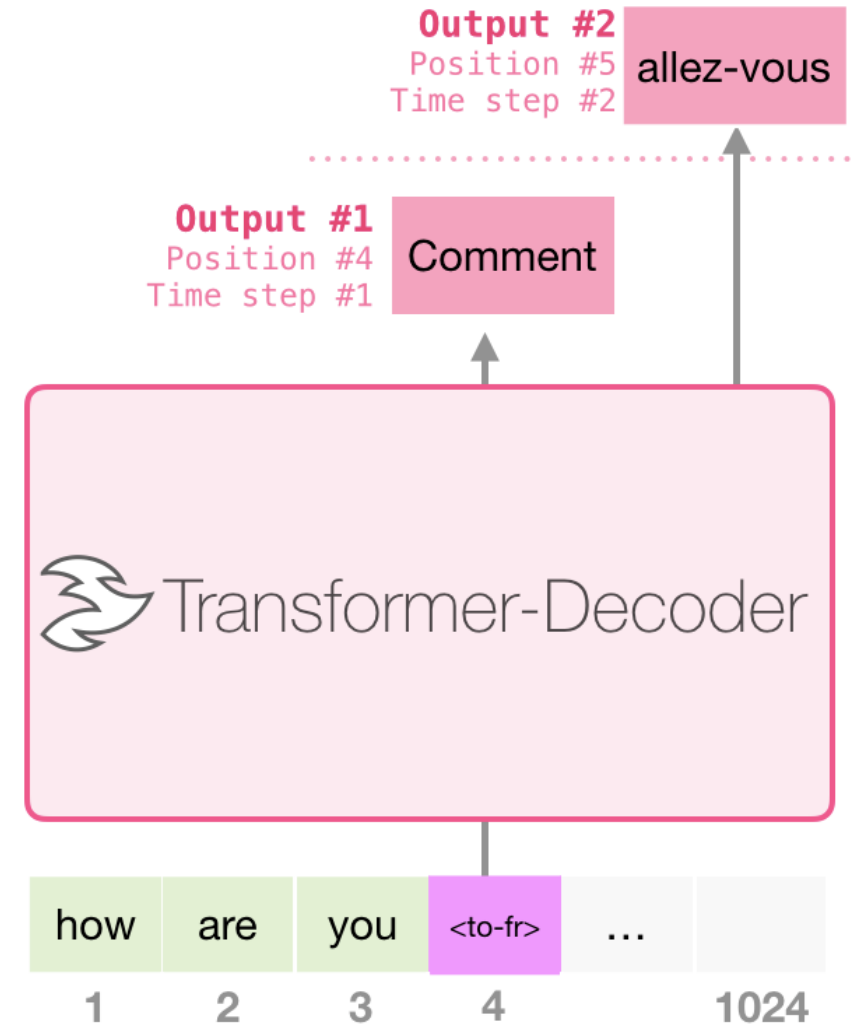
# Video Machine Translation



# Machine Translation based on Generative Models

## Training Dataset

I	am	a	student	<to-fr>	je	suis	étudiant
let	them	eat	cake	<to-fr>	Qu'ils	mangent	de
good	morning	<to-fr>	Bonjour				



# Summarization

# Summarization

WIKIPEDIA

The Free Encyclopedia

Main page

Contents

Featured content

Current events

Random article

Donate to Wikipedia

Wikipedia store

Interaction

Help

About Wikipedia

Community portal

Recent changes

Contact page

Tools

What links here

Related changes

Upload file

Special pages

Permanent link

Page information

Wikidata item

Cite this page

Print/export

Create a book

Download as PDF

Printable version

Languages

English

Français

Italiano

日本語

Português

Svenska

Türkçe

33 more

Edit links

Not logged in

Talk

Contributions

Create account

Log in

Article

Talk

Read

Edit

View history

Search Wikipedia

Positronic brain

From Wikipedia, the free encyclopedia

(Redirected from Positronic robot)

This article is about a fictional technological device. For the manufacturing company based in Springfield, Missouri, see *Positronic* (company).

This article needs additional citations for verification. Please help improve this article by adding citations to reliable sources. Unsourcesd material may be challenged and removed.

Find sources: "Positronic brain" · news · newspapers · books · scholar · JSTOR (July 2008) (Learn how and when to remove this template message)

A **positronic brain** is a fictional technological device, originally conceived by science fiction writer Isaac Asimov.<sup>[1][2]</sup> It functions as a central processing unit (CPU) for robots, and, in some unspecified way, provides them with a form of consciousness recognizable to humans. When Asimov wrote his first robot stories in 1939 and 1940, the positron was a newly discovered particle, and so the buzz word positronic added a contemporary gloss of popular science to the concept. The short story "Runaround", by Asimov, elaborates on the concept, in the context of his fictional Three Laws of Robotics.

Contents [hide]

1

Conceptual overview

2

In Allen's trilogy

3

References in other fiction and films

3.1

Abbott and Costello Go To Mars

3.2

The Avengers

3.3

Doctor Who

3.4

Star Trek

3.5

Perry Rhodan

3.6

(*Robot*, 2004 Film)

3.7

Bicentennial Man

3.8

Buck Rogers in the 25th Century

3.9

Mystery Science Theater 3000

3.10

Spektruman

3.11

Starline

4

References

5

External links

Conceptual overview

[ edit]

Asimov remained vague about the technical details of positronic brains except to assert that their substructure was formed from an alloy of **platinum** and **iridium**. They were said to be vulnerable to radiation and apparently involve a type of *volatile memory* (since robots in storage required a power source keeping their brains "alive"). The focus of Asimov's stories was directed more towards the *software* of robots—such as the Three Laws of Robotics—than the hardware in which it was implemented, although it is stated in his stories that to create a positronic brain without the Three Laws, it would have been necessary to spend years redesigning the fundamental approach towards the brain itself.

Within his stories of *robotics* on Earth and their development by U.S. Robots, Asimov's positronic brain is less of a *plot device* and more of a technological item worthy of study.

A positronic brain cannot ordinarily be built without incorporating the Three Laws; any modification thereof would drastically modify robot behavior. Behavioral dilemmas resulting from conflicting potentials set by inexperienced and/or malicious users of the robot for the Three Laws make up the bulk of Asimov's stories concerning robots. They are resolved by applying the *science* of logic and *psychology* together with *mathematics*, the supreme solution finder being Dr. Susan Calvin, Chief Roboticspsychologist of U.S. Robots.

The Three Laws are also a *bottleneck* in brain sophistication. Very complex brains designed to handle world economy interpret the First Law in expanded sense to include humanity as opposed to a single human; in Asimov's later works like *Robots and Empire* this is referred to as the "Zeroth Law". At least one brain constructed as a calculating machine, as opposed to being a robot control circuit, was designed to have a flexible, childlike personality so that it was able to pursue difficult problems without the Three Laws inhibiting it completely. Specialized brains created for overseeing world economics were stated to have no personality at all.

Under specific conditions, the Three Laws can be obviated, with the modification of the actual robotic design.

- Robots that are of low enough value can have the **Third Law** deleted; they do not have to protect themselves from harm, and the brain size can be reduced by half.
- Robots that do not require orders from a human being may have the **Second Law** deleted, and therefore require smaller brains again, providing they do not require the Third Law.
- Robots that are disposable, cannot receive orders from a human being and are not able to harm a human, will not require even the **First Law**. The sophistication of positronic circuitry renders a brain so small that it could comfortably fit within the skull of an insect.

Robots of the latter type directly parallel contemporary industrial robotics practice, though real-life robots do contain safety sensors and systems, in a concern for human safety (a weak form of the First Law: the robot is a safe tool to use, but has no "judgment", which is implicit in Asimov's own stories).

In Allen's trilogy

[ edit]

Several robot stories have been written by other authors following Asimov's death. For example, in Roger MacBride Allen's *Caliban* trilogy, a Spacer robotist called Gubler Anshaw invents the *gravitronic* brain. It offers speed and capacity improvements over traditional positronic designs, but the strong influence of tradition make robotics labs reject Anshaw's work. Only one robotist, Freda Leving, chooses to adopt gravitronics, because it offers her a blank slate on which she could explore alternatives to the Three Laws. Because they are not dependent upon centuries of earlier research, gravitronic brains can be programmed with the standard Laws, variations of the Laws, or even empty pathways which specify no Laws at all.

WIKIPEDIA

The Free Encyclopedia

Main page

Contents

Featured content

Current events

Random article

Donate to Wikipedia

Wikipedia store

Interaction

Help

About Wikipedia

Community portal

Recent changes

Contact page

Tools

What links here

Related changes

Upload file

Special pages

Permanent link

Page information

Wikidata item

Cite this page

Print/export

Create a book

Download as PDF

Printable version

Languages

English

Français

Italiano

日本語

Português

Svenska

Türkçe

33 more

Edit links

Not logged in

Talk

Contributions

Create account

Log in

Article

Talk

Read

Edit

View history

Search Wikipedia

Positronic brain

From Wikipedia, the free encyclopedia

(Redirected from Positronic robot)

This article is about a fictional technological device. For the manufacturing company based in Springfield, Missouri, see *Positronic* (company).

This article needs additional citations for verification. Please help improve this article by adding citations to reliable sources. Unsourcesd material may be challenged and removed.

Find sources: "Positronic brain" · news · newspapers · books · scholar · JSTOR (July 2008) (Learn how and when to remove this template message)

A **positronic brain** is a fictional technological device, originally conceived by science fiction writer Isaac Asimov.<sup>[1][2]</sup> It functions as a central processing unit (CPU) for robots, and, in some unspecified way, provides them with a form of consciousness recognizable to humans. When Asimov wrote his first robot stories in 1939 and 1940, the positron was a newly discovered particle, and so the buzz word positronic added a contemporary gloss of popular science to the concept. The short story "Runaround", by Asimov, elaborates on the concept, in the context of his fictional Three Laws of Robotics.

Contents [hide]

1

Conceptual overview

2

In Allen's trilogy

3

References in other fiction and films

3.1

Abbott and Costello Go To Mars

3.2

The Avengers

3.3

Doctor Who

3.4

Star Trek

3.5

Perry Rhodan

3.6

(*Robot*, 2004 Film)

3.7

Bicentennial Man

3.8

Buck Rogers in the 25th Century

3.9

Mystery Science Theater 3000

3.10

Spektruman

3.11

Starline

4

References

5

External links

Conceptual overview

[ edit]

Asimov remained vague about the technical details of positronic brains except to assert that their substructure was formed from an alloy of **platinum** and **iridium**. They were said to be vulnerable to radiation and apparently involve a type of *volatile memory* (since robots in storage required a power source keeping their brains "alive"). The focus of Asimov's stories was directed more towards the *software* of robots—such as the Three Laws of Robotics—than the hardware in which it was implemented, although it is stated in his stories that to create a positronic brain without the Three Laws, it would have been necessary to spend years redesigning the fundamental approach towards the brain itself.

Within his stories of *robotics* on Earth and their development by U.S. Robots, Asimov's positronic brain is less of a *plot device* and more of a technological item worthy of study.

A positronic brain cannot ordinarily be built without incorporating the Three Laws; any modification thereof would drastically modify robot behavior. Behavioral dilemmas resulting from conflicting potentials set by inexperienced and/or malicious users of the robot for the Three Laws make up the bulk of Asimov's stories concerning robots. They are resolved by applying the *science* of logic and *psychology* together with *mathematics*, the supreme solution finder being Dr. Susan Calvin, Chief Roboticspsychologist of U.S. Robots.

The Three Laws are also a *bottleneck* in brain sophistication. Very complex brains designed to handle world economy interpret the First Law in expanded sense to include humanity as opposed to a single human; in Asimov's later works like *Robots and Empire* this is referred to as the "Zeroth Law". At least one brain constructed as a calculating machine, as opposed to being a robot control circuit, was designed to have a flexible, childlike personality so that it was able to pursue difficult problems without the Three Laws inhibiting it completely. Specialized brains created for overseeing world economics were stated to have no personality at all.

Under specific conditions, the Three Laws can be obviated, with the modification of the actual robotic design.

- Robots that are of low enough value can have the **Third Law** deleted; they do not have to protect themselves from harm, and the brain size can be reduced by half.
- Robots that do not require orders from a human being may have the **Second Law** deleted, and therefore require smaller brains again, providing they do not require the Third Law.
- Robots that are disposable, cannot receive orders from a human being and are not able to harm a human, will not require even the **First Law**. The sophistication of positronic circuitry renders a brain so small that it could comfortably fit within the skull of an insect.

Robots of the latter type directly parallel contemporary industrial robotics practice, though real-life robots do contain safety sensors and systems, in a concern for human safety (a weak form of the First Law: the robot is a safe tool to use, but has no "judgment", which is implicit in Asimov's own stories).

In Allen's trilogy

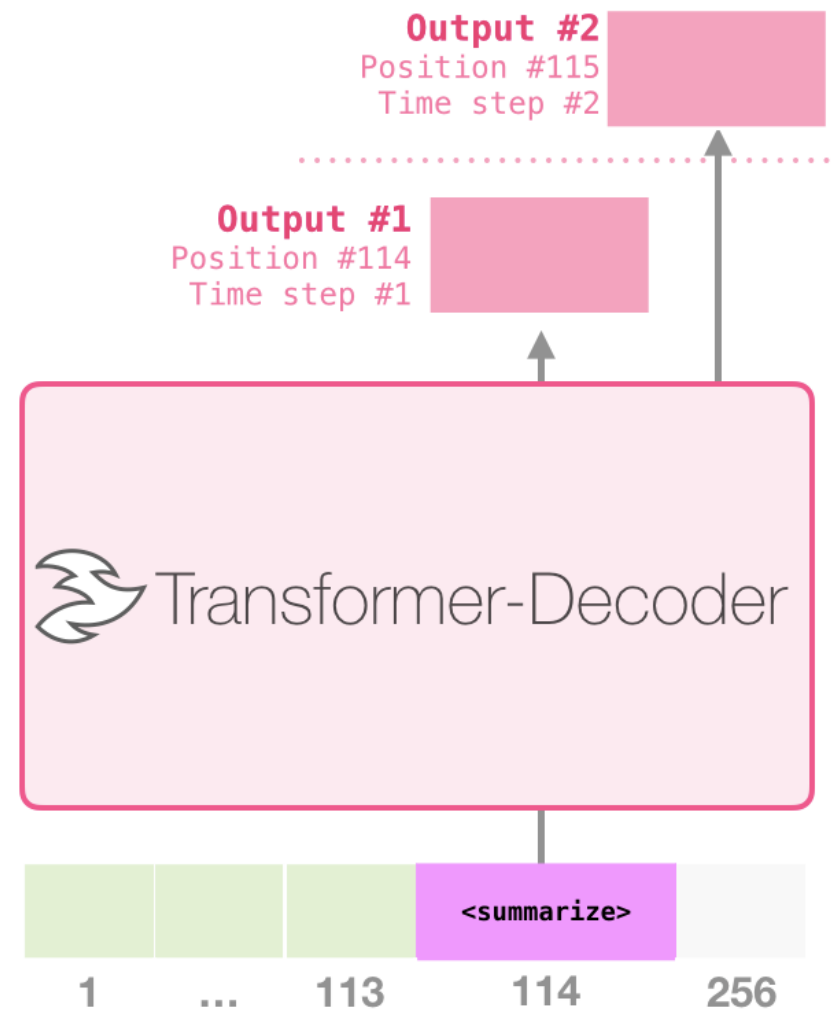
[ edit]

Several robot stories have been written by other authors following Asimov's death. For example, in Roger MacBride Allen's *Caliban* trilogy, a Spacer robotist called Gubler Anshaw invents the *gravitronic* brain. It offers speed and capacity improvements over traditional positronic designs, but the strong influence of tradition make robotics labs reject Anshaw's work. Only one robotist, Freda Leving, chooses to adopt gravitronics, because it offers her a blank slate on which she could explore alternatives to the Three Laws. Because they are not dependent upon centuries of earlier research, gravitronic brains can be programmed with the standard Laws, variations of the Laws, or even empty pathways which specify no Laws at all.

62

## Training Dataset

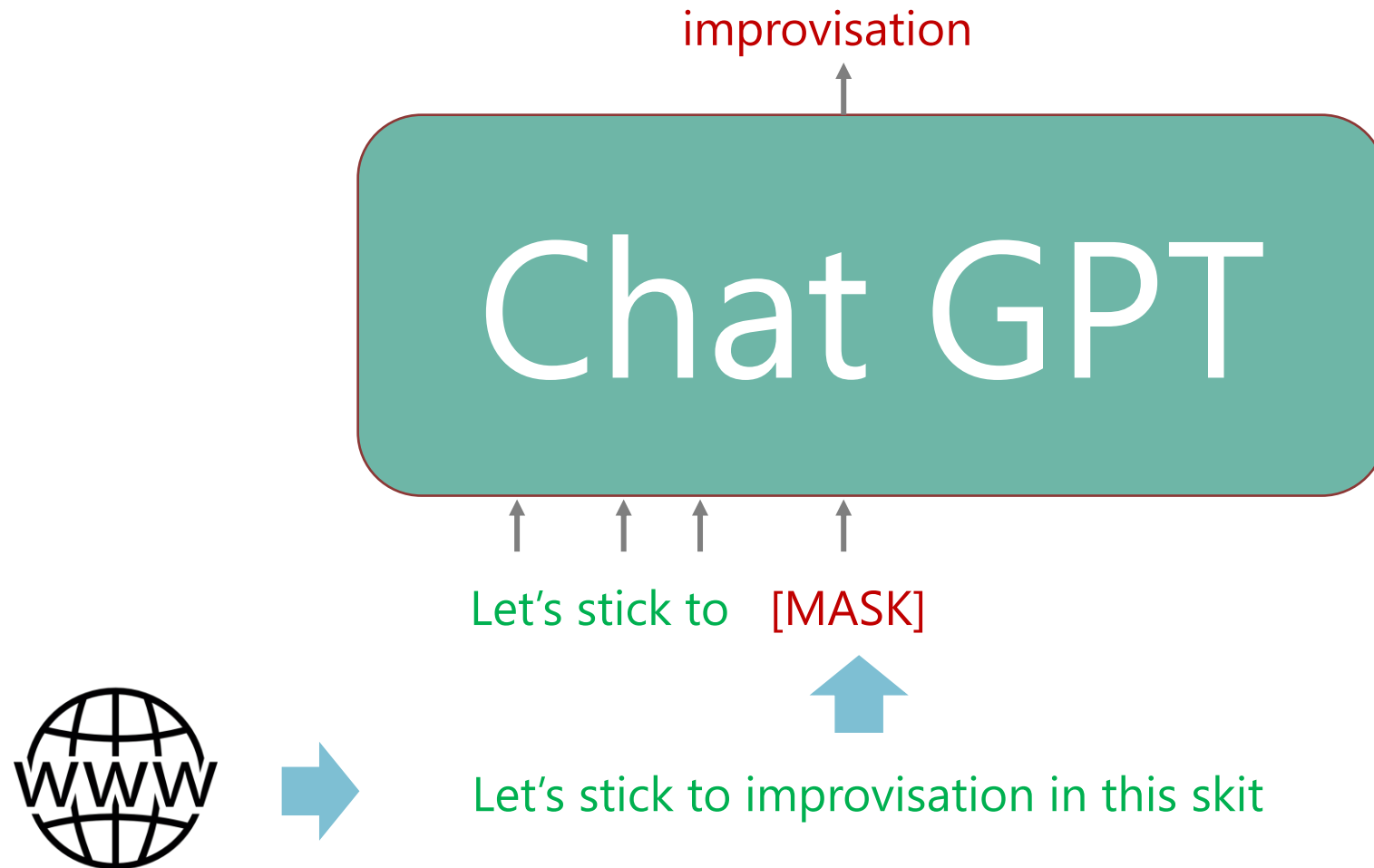
Article #1 tokens	<summarize>	Article #1 Summary
Article #2 tokens	<summarize>	Article #2 Summary
padding		
Article #3 tokens	<summarize>	Article #3 Summary



# Large Language Models

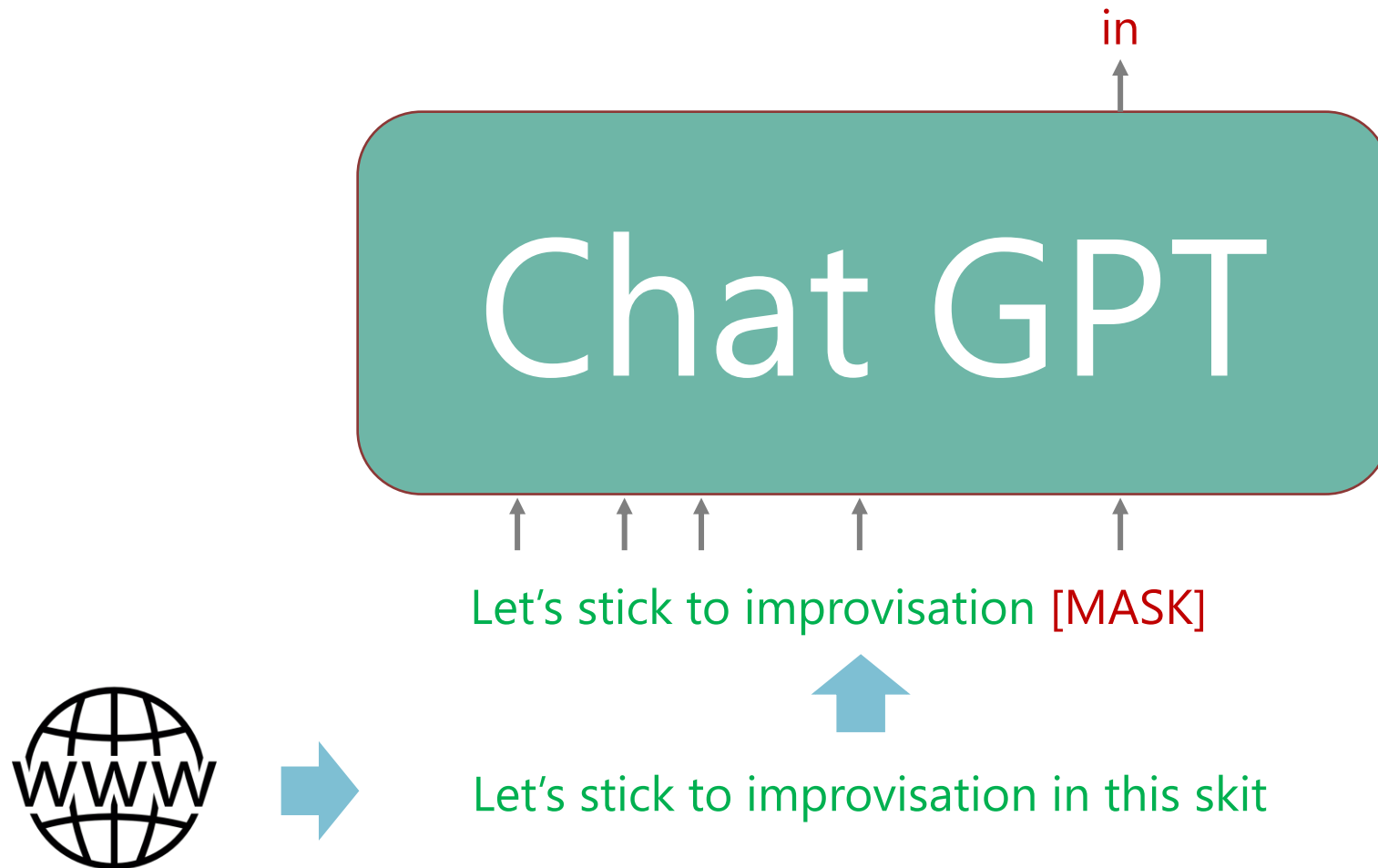


# Learning – Step One: Language Model (Self-Supervised Learning)



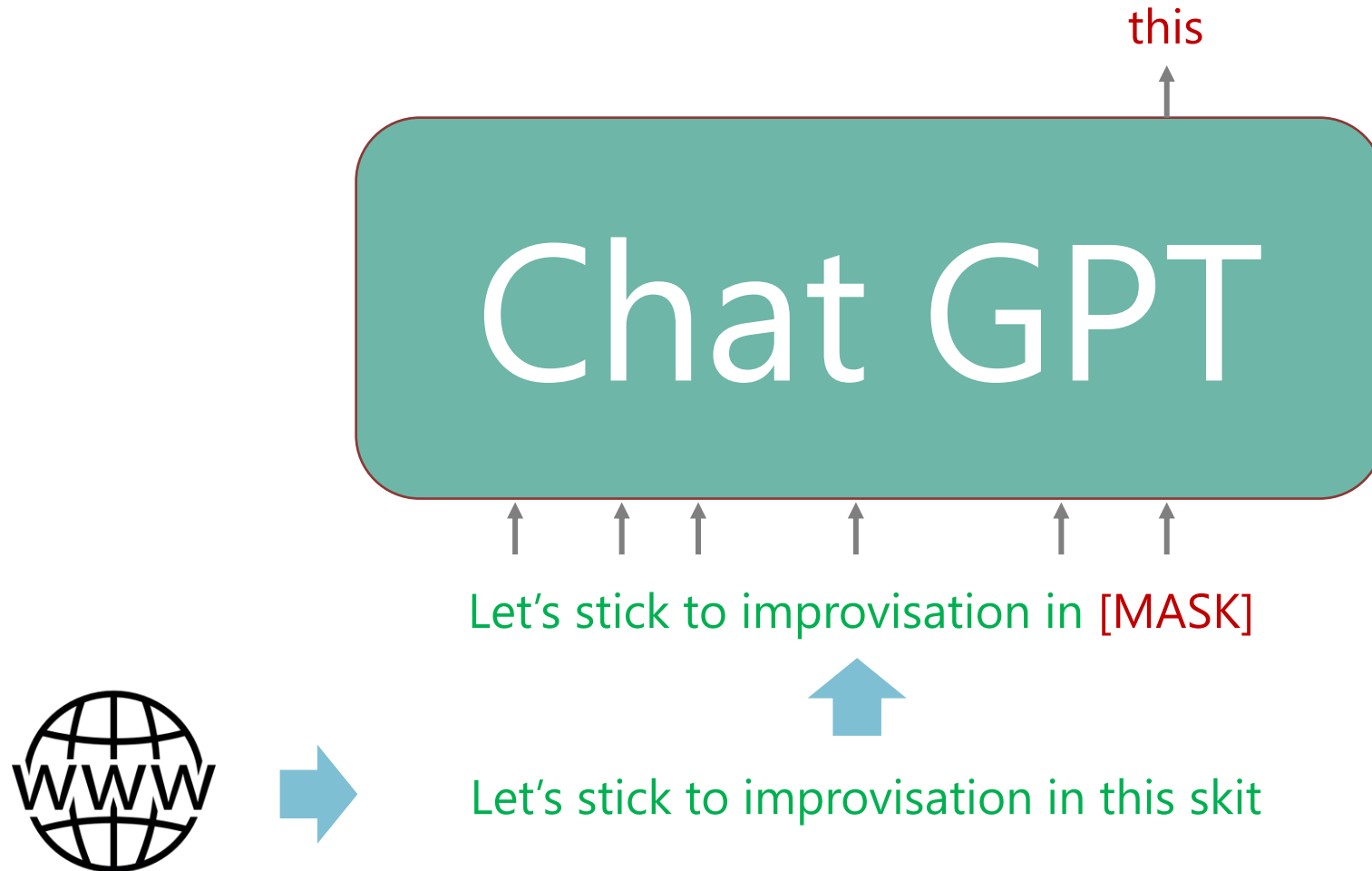
# Learning – Step One: Language Model (Self-Supervised Learning)

---

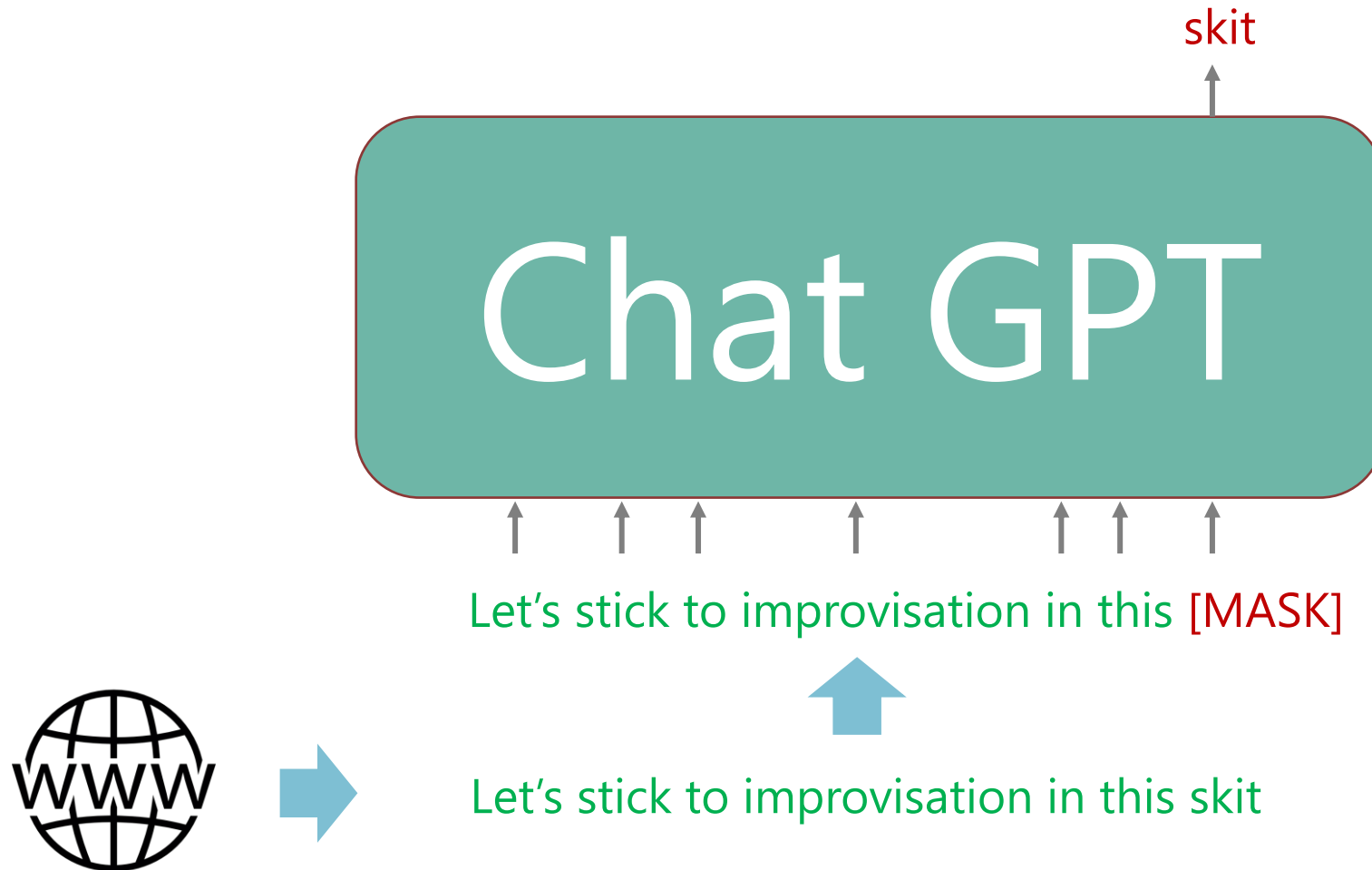


# Learning – Step One: Language Model (Self-Supervised Learning)

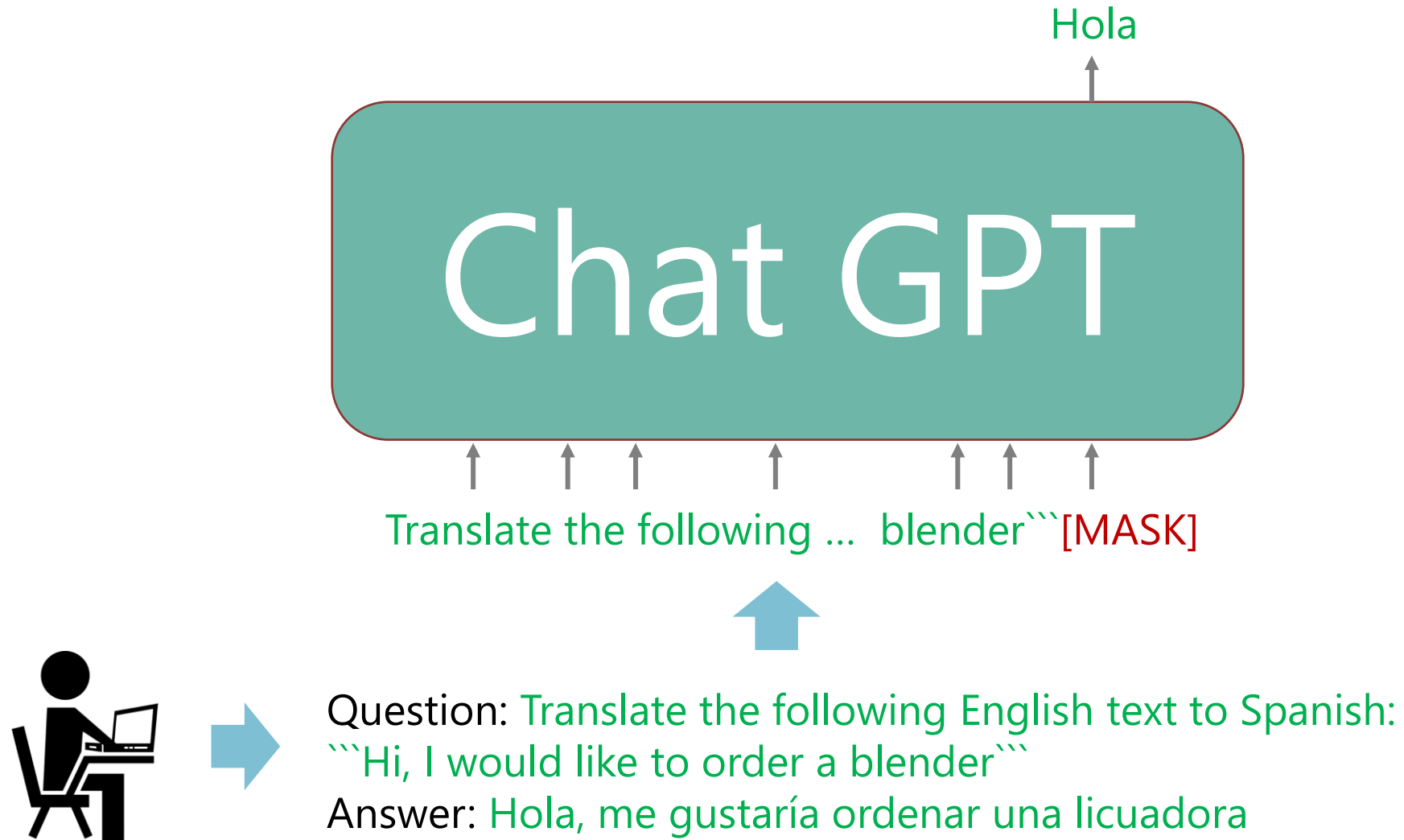
---



# Learning – Step One: Language Model (Self-Supervised Learning)



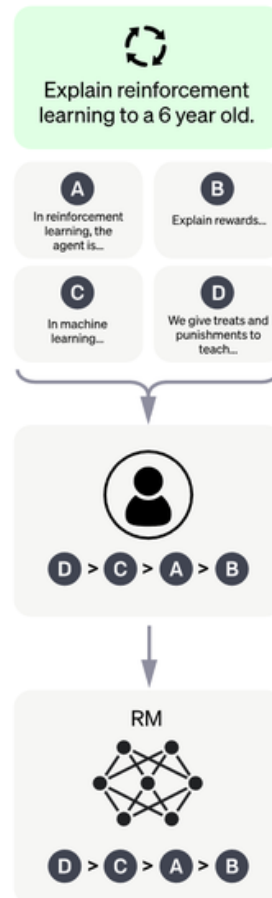
## Learning – Step two: Question Answering (Supervised Learning)



# Learning – Step Three: Reinforcement Learning from Human Feedback (RLHF)

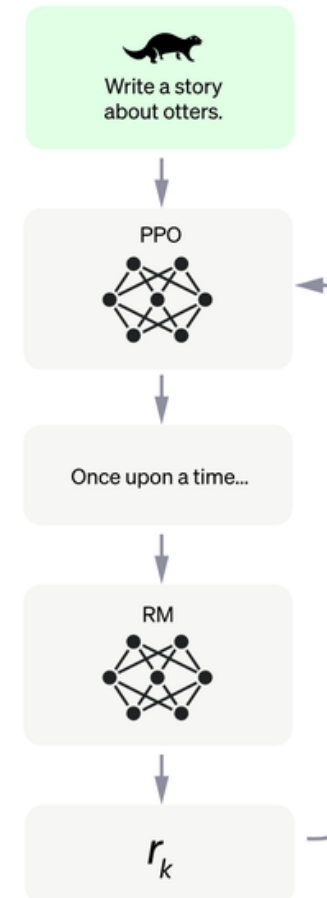
**Collect comparison data and train a reward model.**

A prompt and several model outputs are sampled.



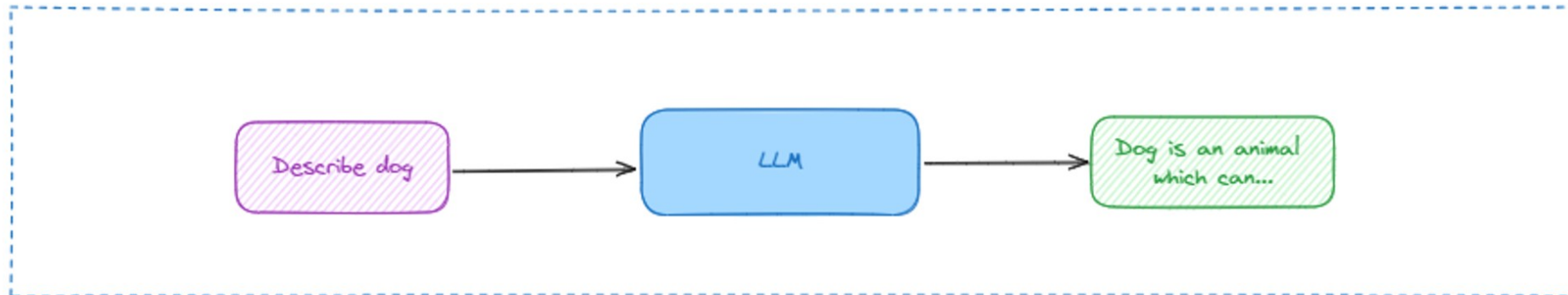
**Optimize a policy against the reward model using the PPO reinforcement learning algorithm.**

A new prompt is sampled from the dataset.

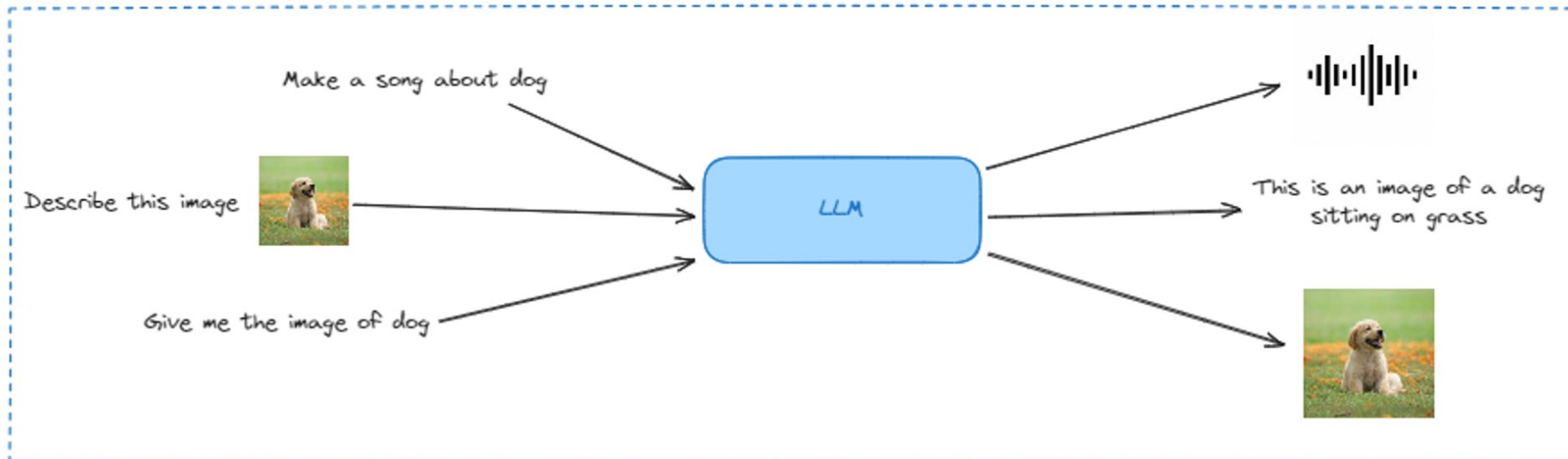


# MultiModal LLMs

## Unimodal Learning



## Multimodal Learning

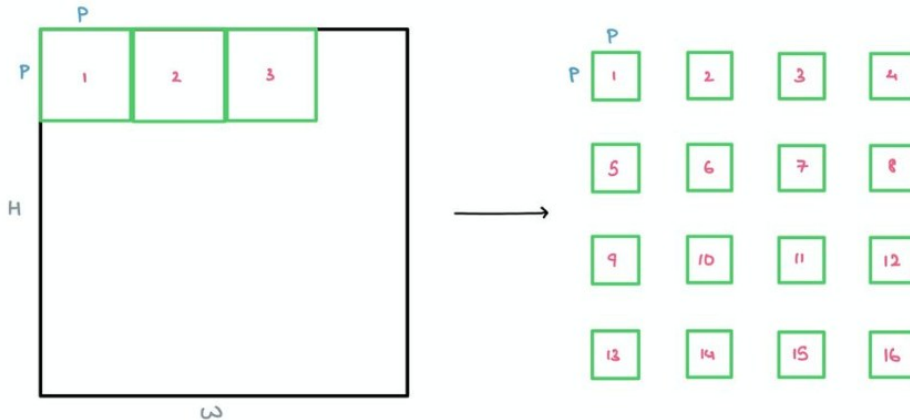


# Vision Transformer

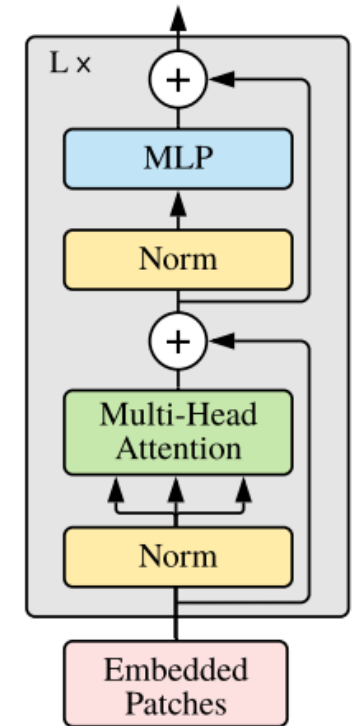
Can the mechanism of attention be applied to images?

Yes, Vision Transformer approach (A. Dosovitskiy, 2020), a model based on the transformer encoder used for image processing (classification)

In ViT, the image is "tokenized" into a patch of  $16 \times 16$  pixels ( $P \times P$  in general)



**Transformer Encoder**





# MultiModal LLMs

